Rapport projet ARA 2017-2018

Mickael Rudek, Oskar Viljasaar

31 janvier 2018

Table des matières

1 Exercice 1 - Implémentation d'un MANET dans PeerSim

1.1 Algorithme de déplacement d'un noeud (Question 1)

```
Movement protocol:
- if not moving
    - assign random speed lower than max
- moving
    - use 'PositioningStrategiesFactory' to get next destination
    - Calc distance to next destination
    - if too far to reach 'destination' in one hop
        - calculate next 'x' and 'y'
        - move to that position
    - if destination reached, stop
    - else continue running
```

L'algorithme utilise le protocole de déplacement suivant : Une valeur de la vitesse est aléatoirement choisie dans l'intervalle [`speed_min`; `speed_max`]. Vu qu'on est en temps discretisé, *distance_to_next* représente la distance parcourue en une unité de temps. Une fois la destination atteinte, le noeud s'arrête pendant un tic, sinon il boucle en demandeant une nouvelle destination de la stratégie de déplacement.

1.2 Contenu du fichier de configuration pour la question 2

```
simulation.endtime 50000
random.seed 5
network.size 10
init.initialisation Initialisation
control.graph GraphicalMonitor
control.graph.positionprotocol position
control.graph.time_slow 0.0002
control.graph.step 1
```

1.3 Questions 3 et 4

Voir la sous-section 1.6.

1.4 Implémentation de l'interface `Emitter` (Question 5)

1.5 Influence des stratégies sur la connexité du graphe (Questions 3, 4, 8)

Strategy1InitNext donne des positions initiales et destinations aléatoires dans le terrain pour chaque noeud.
Strategy3InitNext donne des positions initiales et destinations vers le milieu du terrain, dans un rayon de `scope` − `marge`, assurant un graphe connexe.
Strategy2Next rend les noeuds immobiles, la connexité du graphe dépend du placement initial des noeuds.
Strategy4Next assume que le graphe est connexe à l'initiation. Elle va déplacer un nœud dans le graphe en s'assurant qu'à la fin, le graphe soit toujours connexe. La connexité du graphe dépend du placement initial des noeuds.
Strategy5Init place les noeuds en haut à droite du terrain, chaque noeud est placé dans le scope d'un autre noeud. Le graphe est initialement connexe.
Strategy6Init place les noeuds en étoile au milieu du terrain, le graphe est donc initialement connexe.

SPI	*SD*	**Connexe**
Strategy1InitNext	Strategy1InitNext	non
Strategy1InitNext	Strategy2Next	non
Strategy1InitNext	**Strategy3InitNext**	**oui**
Strategy1InitNext	Strategy4Next	non
Strategy3InitNext	Strategy1InitNext	non
Strategy3InitNext	Strategy2Next	**oui**
Strategy3InitNext	**Strategy3InitNext**	**oui**
Strategy3InitNext	Strategy4Next	**oui**
Strategy5Init	Strategy1InitNext	non
Strategy5Init	Strategy2Next	**oui**
Strategy5Init	**Strategy3InitNext**	**oui**
Strategy5Init	Strategy4Next	**oui**
Strategy6Init	Strategy1InitNext	non
Strategy6Init	Strategy2Next	**oui**
Strategy6Init	**Strategy3InitNext**	**oui**
Strategy6Init	Strategy4Next	**oui**

TABLE 1 – Impact des différentes SPI et SD sur la connexité du graphe

1.6 Impact de la portée sur la densité du graphe (Questions 10 et 11)

Dans la stratégie 1, l'étendue de la portée a un impact sur la connexité du graphe, la stratégie de déplacement étant celle de choisir des destinations aléatoires dans le terrain. Il est plus facile donc de faire un graphe connexe en prenant une valeur assez grande pour la portée.
La stratégie 3 donnant un graphe connexe dès le début, les noeuds disposent déjà d'un nombre de voisins important. Augmenter la portée pour la stratégie 3 a tendance à légèrement faire diminuer la densité du graphe. Cela peut être expliqué par la distance aléatoire pour la prochaine destination, tirée entre `NextDestinationStrategy.minimum_distance` et $scope - marge$, sachant que $marge$ est plutôt petit (20) et reste constant, alors que la portée peut varier jusqu'à 1000. Le graphe, dans la stratégie 3, est beaucoup plus étendu, et les noeuds peuvent avoir moins d'arcs directs entre eux.

Portee	SPI	SD	D	E/D	ED/D
125	1	1	1.00 +- 0.02	0.27 +- 0.02	0.04 +- 0.00
250	1	1	3.81 +- 0.10	0.14 +- 0.00	0.04 +- 0.00
375	1	1	8.02 +- 0.20	0.13 +- 0.02	0.09 +- 0.02
500	1	1	12.83 +- 0.06	0.11 +- 0.02	0.09 +- 0.02
625	1	1	18.77 +- 0.54	0.11 +- 0.00	0.13 +- 0.01
750	1	1	24.49 +- 0.13	0.10 +- 0.00	0.16 +- 0.02
875	1	1	29.92 +- 0.47	0.09 +- 0.00	0.16 +- 0.02
1000	1	1	35.66 +- 0.44	0.07 +- 0.01	0.11 +- 0.04
125	3	3	29.94 +- 0.25	0.09 +- 0.00	0.17 +- 0.02
250	3	3	26.78 +- 0.27	0.10 +- 0.01	0.21 +- 0.05
375	3	3	25.82 +- 0.41	0.09 +- 0.00	0.16 +- 0.01
500	3	3	25.75 +- 0.58	0.10 +- 0.00	0.15 +- 0.02
625	3	3	25.52 +- 0.33	0.09 +- 0.00	0.15 +- 0.03
750	3	3	25.80 +- 0.23	0.10 +- 0.00	0.16 +- 0.02
875	3	3	25.61 +- 0.47	0.10 +- 0.00	0.16 +- 0.02
1000	3	3	25.21 +- 0.31	0.10 +- 0.00	0.16 +- 0.02

TABLE 2 – Valeurs obtenues pour la question 10, normalisées sur 100 itérations

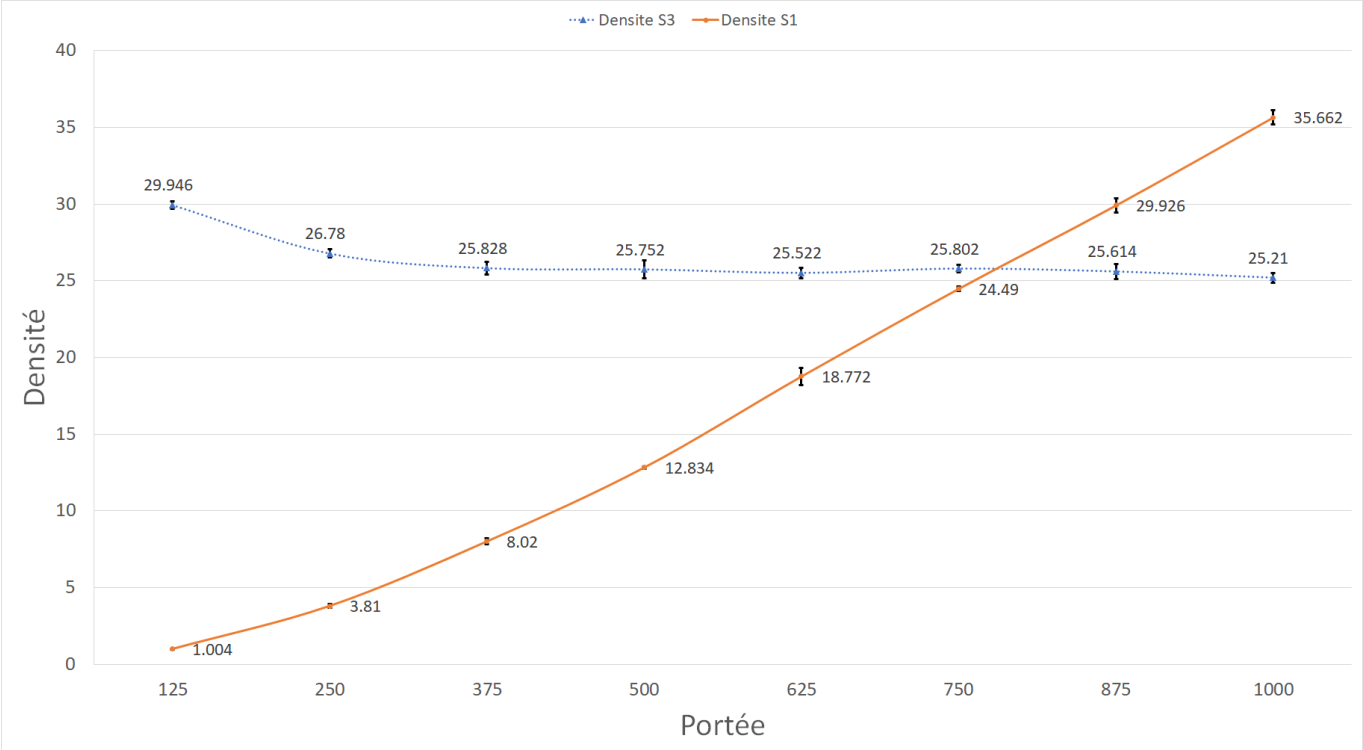

FIGURE 1 – Impact de la portée sur la densité avec la stratégie 1 (orange) et la 3 (bleu).

2 Exercice 2 - Étude de protocoles de diffusion

2.1 Impact du nombre de noeuds sur la densité du graphe (Question 1)

Selon le graphe, le réseau est plutôt chaotique sur un nombre de noeuds faible, allant jusqu'à 50 selon nos résultats. L'écart-type entre les différentes valeurs mesurées est fort, le réseau peut avoir du mal à s'interconnecter selon les différentes valeurs initiales aléatoires prises. À partir de 50 noeuds, le réseau a un comportement attendu et la densité croît de manière relativement stable, avec un écart-type faible entre les mesures.
En dessous de 50 noeuds, on ne peut donc pas forcément s'attendre à un placement de noeuds idéal, de manière à ce que tous les noeuds soient connectés avec beaucoup de voisins à proximité.

Taille	D-end	ED/D end
10	3.32 +- 0.00	0.26 +- 0.00
20	6.88 +- 1.03	0.55 +- 0.30
30	10.92 +- 2.11	0.28 +- 0.20
40	16.04 +- 4.00	0.17 +- 0.11
50	21.95 +- 4.95	0.08 +- 0.01
60	23.37 +- 6.02	0.13 +- 0.05
70	23.11 +- 4.74	0.08 +- 0.03
80	27.26 +- 1.76	0.07 +- 0.03
90	37.59 +- 3.83	0.06 +- 0.02
100	35.38 +- 3.23	0.04 +- 0.01
120	37.78 +- 5.27	0.03 +- 0.01
140	47.42 +- 12.2	0.04 +- 0.01
160	49.15 +- 10.2	0.03 +- 0.02
180	62.37 +- 5.31	0.03 +- 0.01
200	50.33 +- 7.25	0.02 +- 0.01

TABLE 3 – Valeurs obtenues pour la question 1, normalisés sur 100 itérations

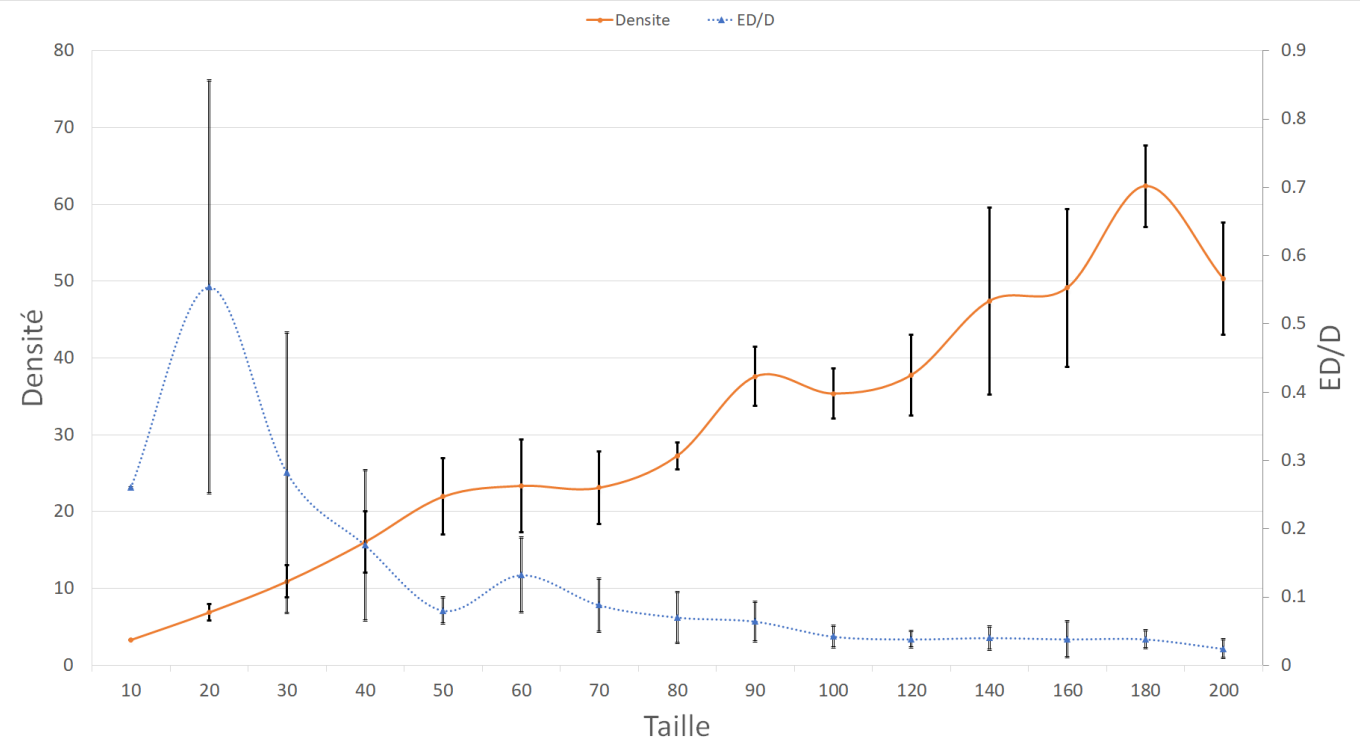

FIGURE 2 – Impact du nombre de noeuds sur la densité et sa variation possible pendant une simulation, avec les stratégies $SPI5$ et $SD4$.

2.2 Question 2 (`EmitterCounter`)

On a défini une classe abstraite `EmitterCounter` utilisant le design pattern *Strategy*, en rendant la méthode `emit` abstraite. Une sous-classe concrète de celle-ci implémente une politique particulière d'émission (`FloodingEmitter`, `ProbabilisticEmitter`, ...) et rend le nombre de messages envoyés selon cette politique. Cette information est destinée à `GossipProtocol` qui se charge de la délivrance (ou non) du message selon si le noeud émetteur se trouve encore dans la portée du récepteur.

On a essayé d'implémenter cette politique de délivrance au niveau de l'émetteur (`EmitterCounter`) en le faisant traiter des réceptions de messages de son protocole encapsulant des messages de n'importe quel protocole au-dessus de lui-même, mais le simulateur rendait trop difficile de détecter une terminaison de manière simple. La fonction EDSimulator.add(0, ..) rajoute un évènement à la fin de la file d'éxecution, mais on voulait faire un traitement juste après la délivrance du message par `EmitterCounter`.

2.3 Question 4 (`FloodingEmitter`)

Quelque soit la densité du graphe, tant qu'il est connexe, `FloodingEmitter` assure une atteignabilité de diffusion de 100%. L'économie de rediffusion est naturellement nulle, comme l'émetteur effectue une rediffusion dans tous les cas.

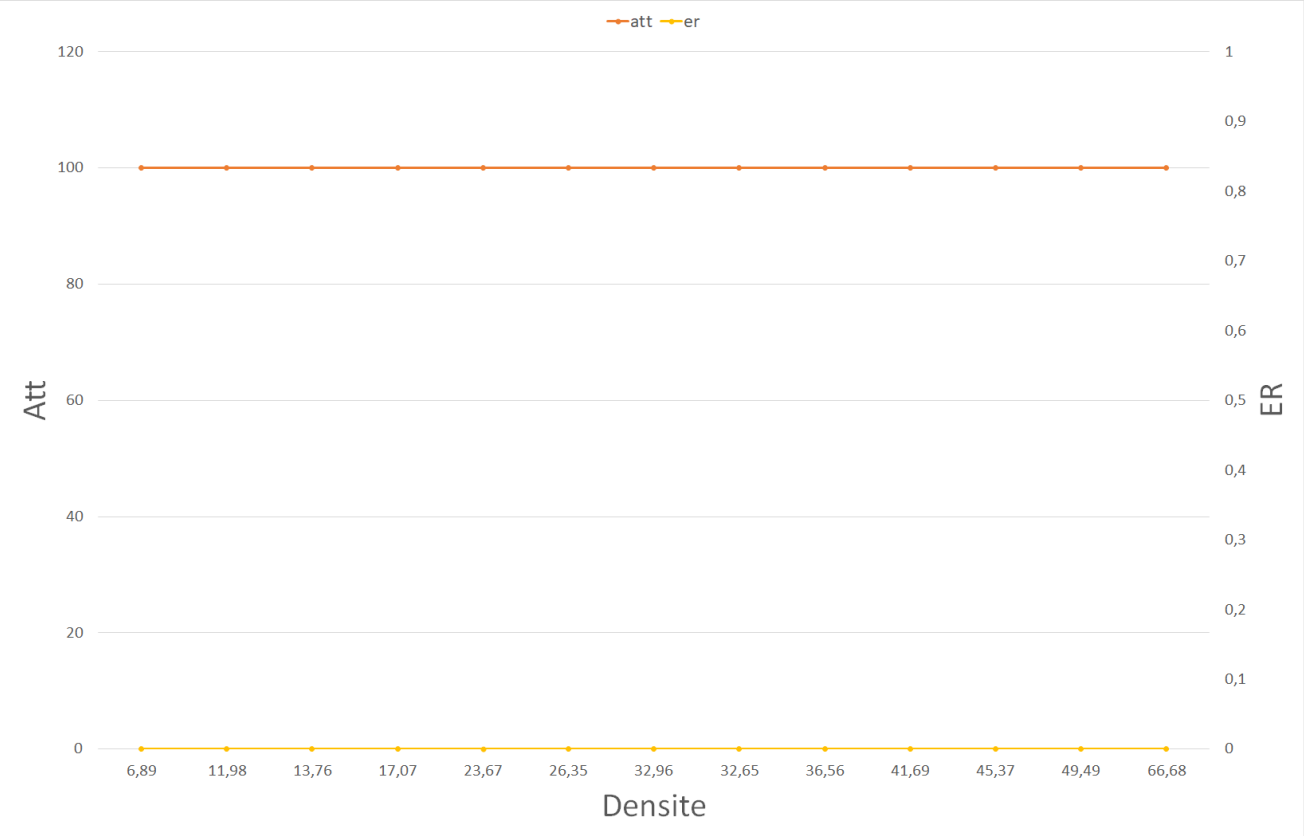

FIGURE 3 – Atteignabilité et économie de rediffusion avec `FloodingEmitter`.

2.4 Question 5 (`ProbabilisticEmitter`)

Le pourcentage de messages reçus augmente clairement selon la probabilité. Selon les résultats expérimentaux, il faut définir une probabilité autour de 0.3 afin d'obtenir une atteignabilité d'au moins 90%. À partir d'une densité de graphe de 5, l'atteignabilité ne descend pas en-dessous des 80%. Pour obtenir une atteignabilité moyenne du graphe d'au moins 99%, il faudrait utiliser une probabilité d'au moins 0.7.

Les courbes figurant sur le graphe représentent les différentes classes d'atteignabilité définies dans la question de l'exercice. La courbe grise et orange montrent qu'il est vite possible d'atteindre la majorité du graphe, même avec une densité faible.

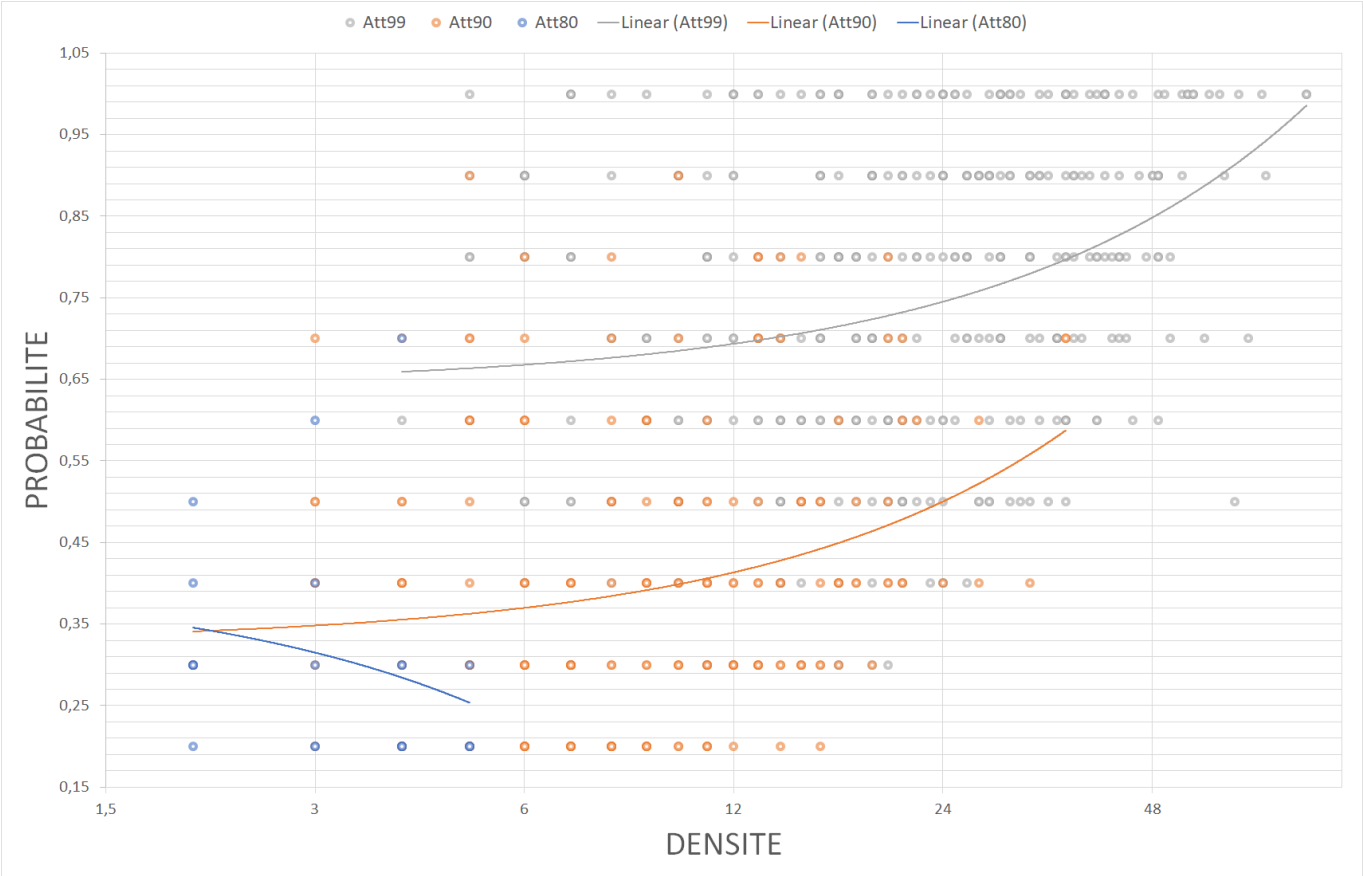

FIGURE 4 – Probabilité en fonction de la densité. Le graphe contient les valeurs de 5 éxecutions différentes pour un même ensemble de valeurs initiales (nombre de noeuds/densité et probabilité).

2.5 Question 6 (`InverseProportionalEmitter`)

2.6 Question 7 (`DistanceEmitter`)

A Compilation, lancement du code et jeux de test

A.1 src/Makefile

Un fichier `Makefile` est inclus dans le dossier `src`. Celui-ci reconnaît les directives :
— `make` compile le projet.
— `make run` lance une instance de simulation telle que spécifiée dans `src/manet/cfg_initial.txt`.
— `make clean` nettoie le projet des compilés.
— `make bench_clean` nettoie le dossier `src` des dossiers de résultats des benchmarks.

Le `Makefile` admet par ailleurs deux variables :
— `DIR_PEERSIM=<chemin>` : le dossier d'installation de Peersim, qu'il faudra **obligatoirement** soit modifier, soit spécifier dans `make` et `make run`.
— `CFG=<chemin>` : le chemin d'un fichier de configuration Peersim, initialisé à `src/manet/cfg_initial.txt` par défaut.

A.2 src/bench.pl

Exemple : `./bench.pl <chemin_peersim>`
Le script crée un dossier `resultats/bench_<date>` où seront stockés les résultats pour la question 8 de l'exercise 1. Le dossier contiendra les fichiers de configuration pour les expériences sous le nom `cfg_bench_<scope>_<SPI>_<SD>`, les résultats d'autres fichiers, de même nom avec l'extension `.result`. Ces derniers contiennent les résultats de 100 expériences avec autant de différentes graines aléatoires.

A.3 src/bench.py

exemple : `bench.py <chemin_results>`
Dans le dossier `<chemin_results>` doivent se trouver les fichiers résultats créés par `bench.pl`. Utilisé pour les resultats de benchmark, pour remplir les tableaux.
Exemple de sortie :

```
/ara/src/results/cfg_bench_500_3_3.result
| 2.3830 +- 0.0684 | 0.4320 +- 0.0426 | 0.2300 +- 0.0335 |

/ara/src/results/cfg_bench_875_3_3.result
| 2.3390 +- 0.1023 | 0.4400 +- 0.0369 | 0.2140 +- 0.0201 |

/ara/src/results/cfg_bench_750_1_1.result
| 2.2740 +- 0.1165 | 0.4630 +- 0.0518 | 0.2320 +- 0.0325 |

/ara/src/results/cfg_bench_375_1_1.result
| 0.7130 +- 0.0447 | 0.6550 +- 0.0457 | 0.1790 +- 0.0230 |
```

Les valeurs correspondent à des moyennes avec écarts-type, obtenus sur 100 itérations sur chaque combinaison de SPI et SD.

A.4 src/bench_ex2.py

exemple : `bench_ex2.py <chemin_results>`
Ce script traite les resultats des simulations de l'exercise deux. Il prepare deux fichiers, stockes dans le dossier specifie dans l'appel - `summary.csv` et `summary_all.csv`.

`summary.csv` contient des valeurs normalisées sur le nombre d'expériences, tandis que `summary_all.csv` stocke simplement tous les résultats, sans compter les moyennes.

A.5 bench-ex2q{1,3,4}.pl

Exemple : `./bench-ex2q{1,3,4}.pl <chemin_peersim>`
Le script crée un dossier `resultats/bench_<date>_ex2q{1,3,4}` où seront stockés les résultats des différentes exécutions, les noms des fichiers étant différentiés par la taille du réseau.

A.6 bench-ex2q5.pl

Exemple : `./bench-ex2q5.pl <chemin_peersim>` Le script crée un dossier `resultats/bench_<date>_ex2q5` où seront stockés les résultats des différentes exécutions, les noms des fichiers étant différentiés par la taille du réseau, la probabilité donnée et le numéro de l'itération.

Les scripts de l'exercice 2 (hormis la question 1) sortent un fichier par exécutions, de forme :

```
Att;Ed-Att;ER;Ed-ER
D;ED;ED/D
```

B Extraits de code

B.1 Implémentation de l'interface `Emitter` (Exercice 1 Question 5)

```
1  package manet.communication;
2
3  import manet.Message;
4  import manet.positioning.Position;
5  import manet.positioning.PositionProtocol;
6  import peersim.config.Configuration;
7  import peersim.core.Network;
8  import peersim.core.Node;
9  import peersim.core.Protocol;
10 import peersim.edsim.EDSimulator;
11
12 public class EmitterImpl implements Emitter {
13
14     private int latency;
15     private int scope;
16     private int this_pid;
17     private int position_protocol;
18
19     private static final String PAR_LATENCY = "latency";
20     private static final String PAR_SCOPE = "scope";
21     private static final String PAR_POSITIONPROTOCOL = "positionprotocol";
22
23     public EmitterImpl(String prefix) {
24         String tmp[]=prefix.split("\\.");
25         this_pid=Configuration.lookupPid(tmp[tmp.length-1]);
26         this.position_protocol=Configuration.getPid(prefix+"."+PAR_POSITIONPROTOCOL);
27         this.latency = Configuration.getInt(prefix + "." + PAR_LATENCY);
28         this.scope = Configuration.getInt(prefix + "." + PAR_SCOPE);
29     }
30
31     @Override
32     public void emit(Node host, Message msg) {
33         PositionProtocol prot =
34             (PositionProtocol) host.getProtocol(position_protocol);
35
36         for (int i=0; i < Network.size(); i++) {
37             Node n = Network.get(i);
38             PositionProtocol prot2 =
39                 (PositionProtocol) n.getProtocol(position_protocol);
40             double dist =
41                 prot.getCurrentPosition().distance(prot2.getCurrentPosition());
42             if (dist < scope && n.getID() != host.getID()) {
43                 EDSimulator.add(latency, new Message(msg.getIdSrc(),
44                                                      n.getID(),
45                                                      msg.getTag(),
46                                                      msg.getContent(),
47                                                      msg.getPid()),
48                                                      n, msg.getPid());
49             }
```

```
        }

    }

    @Override
    public int getLatency() { return latency; }

    @Override
    public int getScope() { return scope; }

    @Override
    public Object clone(){
        EmitterImpl res=null;
        try {
            res=(EmitterImpl)super.clone();
        } catch (CloneNotSupportedException e) {}
        return res;
    }
}
```

B.2 Implémentation de l'interface `NeighborProtocol` (Exercice 1 Question 6)

```
package manet.detection;

import manet.Message;
import manet.communication.EmitterImpl;
import peersim.config.Configuration;
import peersim.core.Node;
import peersim.edsim.EDProtocol;
import peersim.edsim.EDSimulator;

import java.util.ArrayList;
import java.util.List;

public class NeighborProtocolImpl implements NeighborProtocol, EDProtocol {
    private int this_pid;
    private int period;
    private int timer_delay;
    private int listener_pid;

    private static final String PAR_PERIOD = "period";
    private static final String PAR_TIMERDELAY = "timer_delay";
    private static final String PAR_LISTENER_PID = "listenerpid";
    Integer timeStamp = 0;

    private List<Long> neighbor_list;

    public NeighborProtocolImpl(String prefix) {
        neighbor_list = new ArrayList<>();

        String tmp[]=prefix.split("\\.");
        this_pid= Configuration.lookupPid(tmp[tmp.length-1]);
        this.period = Configuration.getInt(prefix+"."+PAR_PERIOD);
        this.timer_delay = Configuration.getInt(prefix + "." + PAR_TIMERDELAY);
        this.listener_pid = Configuration.getPid(prefix + "." + PAR_LISTENER_PID,-1);
        }

    @Override
    public List<Long> getNeighbors() { return neighbor_list; }

    @Override
    public Object clone() {
        NeighborProtocolImpl res = null;
```

```
        try {
            res = (NeighborProtocolImpl) super.clone();
            neighbor_list = new ArrayList<>();
            timeStamp = new Integer(0);
        } catch (CloneNotSupportedException e) {

        }
        return res;
    }

    @Override
    public void processEvent(Node node, int pid, Object event) {
        int emitter_pid = Configuration.lookupPid("emitter");
        EmitterImpl impl = (EmitterImpl) node.getProtocol(emitter_pid);
        Message msg = (Message) event;

        if (event instanceof Message) {
            switch (msg.getTag()) {
                case "Heartbeat":
                    if (msg.getIdSrc() == msg.getIdDest()) {
                        EDSimulator.add(this.period, event, node, pid);
                        impl.emit(node, new Message(node.getID(), 0,
                                        "Heartbeat",
                                        "Heartbeat", this_pid));
                    }
                    else {
                        if(!neighbor_list.contains(msg.getIdSrc()))
                            neighbor_list.add(msg.getIdSrc());
                        break;
                    }
                    break;
                default:
                    System.out.println("IN DEFAULT");
            }
        }
        else {
            System.out.println("no good message");
        }
        return;
    }
}
```

B.3 DensityController (Exercice 1 Question 9)

```
package manet;

import manet.detection.NeighborProtocol;
import peersim.config.Configuration;
import peersim.core.CommonState;
import peersim.core.Control;
import peersim.core.Network;

import java.io.FileOutputStream;
import java.util.ArrayList;

public class DensityController implements Control {


    private static final String PAR_NEIGHBOR = "neighbours";
    private static final String PAR_VERBOSE = "verbose";
    private static final String PAR_STEP = "step";

    private final int this_pid;
```

```
20
21      private int verbose = 0;
22      // Est-ce que l'on print les resultats sur stdout
23      private int step; // step du controlleur
24
25      private double
26              dit = 0.0,  // la moyenne du nombre de voisins par noeud a l'instant t (dens
27              eit = 0.0,  // l'ecart-type de dit (donc a l'instant t)
28              dt  = 0.0,  // densite moyenne sur le temps (avg of d_dt)
29              et  = 0.0,  // disparite moyenne de densite sur le temps (avg of d_et)
30              edt = 0.0;  // variation de la densite au cours du temps (ecart type des val
31      // donc de toute la sim jusqu'a mtn)
32
33      // Arrays containing data for dt, et and edt calculations
34      private ArrayList<Double>
35              d_dt  = new ArrayList<Double>(), // Updated by dit()
36              d_et  = new ArrayList<Double>(), // Updated by eit()
37              d_edt = new ArrayList<Double>(); // Updates by edt()
38
39      public DensityController(String prefix) {
40          this.this_pid = Configuration.getPid(prefix+"."+PAR_NEIGHBOR);
41          this.verbose = Configuration.getInt(prefix + "." + PAR_VERBOSE);
42          this.step = Configuration.getInt(prefix + "." + PAR_STEP);
43      }
44
45
46      @Override
47      // @return true if the simulation has to be stopped, false otherwise.
48      public boolean execute() {
49          // Over-time averages
50          dt = dt();
51          et = et();
52          edt = edt();
53
54          // 'Live' values
55          dit = dit();
56          eit = eit();
57
58          if (this.verbose != 0)
59              if (CommonState.getTime() >= CommonState.getEndTime()-step)
60                  printCols();
61
62          return false;
63      }
64
65      /** A l'instant T **/
66
67      /**
68       * Calculates the average number of neighbours in the
69       * network when called (works on 'live' data)
70       * D_i(t) : Moyenne du nombre de voisins par noeud a l'instant t
71       *
72       * Updates dit and d_dt[]
73       *
74       * @return double average neighbors per node
75       */
76      private double dit() {
77          double
78                  sum = 0.0,
79                  avg = 0.0;
80
81          for (int i = 0 ; i < Network.size() ; i++) {
82              double n_neigs = ((NeighborProtocol) Network.get(i).getProtocol(this_pid)).g
```

```
83              sum += n_neigs;
84          }
85
86          avg = sum / Network.size();
87          d_dt.add(avg);  // Add to history
88          return avg;
89      }
90
91      /**
92       * Calculates the standard deviation
93       * E_i(t) : L'ecart type de D_i(t) (dit())
94       * Works on 'live' data
95       * Updates eit and d_et[]
96       *
97       * @return l'ecart-type de dit
98       */
99      public double eit() {
100         double stdDev = 0.0;
101         for (Double d : d_dt) {
102             if (this.verbose != 0)
103                 System.out.format("d: %.2f\n", d);
104             stdDev += Math.pow(d - dit, 2);
105         }
106         if (this.verbose != 0)
107             System.out.format("Stddev: %.2f ", stdDev);
108         double avg = stdDev/d_dt.size();
109         stdDev = Math.sqrt(avg);
110         if (this.verbose != 0)
111             System.out.format("avg: %.2f stddev: %.2f\n", avg, stdDev);
112         d_et.add(stdDev);   // Add to history
113         return stdDev;
114     }
115
116
117
118     /** Stats for all until current **/
119
120     /**
121      * La moyenne de l'ensemble des valeurs D_i(t') pour tout t' < t
122      * donc densite moyenne sur le temps
123      *
124      * Updates dt, works with history array
125      *
126      * @return average density so far
127      */
128     public double dt() {
129         double avg = 0.0;
130         if (!d_dt.isEmpty()) {
131             for (Double d : d_dt)
132                 avg += d;
133             avg = avg / d_dt.size();
134         }
135         return avg;
136     }
137
138     /**
139      * La moyenne de l'ensemble des valeurs E_i(t') pour tout t' < t
140      * donc disparite moyenne de densite sur le temps
141      *
142      * Updates et, works with history array
143      *
144      * @return average density so far
145      */
```

```
146     public double et() {
147         double avg = 0.0;
148         if (!d_et.isEmpty()) {
149             for (Double d : d_et)
150                 avg += d;
151             avg = avg / d_et.size();
152         }
153         return avg;
154     }
155 
156     /**
157      * L'ecart type des valeurs D_i(t'), pour tout t' <= t, ce qui
158      * permet de juger de la variation de la densite au cours du temps.
159      * Plus le @return de cette fonction est elevee par rapport au resultat
160      * de et(), plus le reseau a change de densite moyenne au cours
161      * du temps.
162      *
163      * Updates recalculates etd, works with history array
164      *
165      * @return
166      */
167     public double edt() {
168         double stdDev = 0.0;
169         if (!d_dt.isEmpty()) {
170             for (Double d : d_dt)
171                 stdDev += Math.pow(dt - d, 2);
172             stdDev = stdDev / d_dt.size();
173         }
174         d_edt.add(stdDev);
175         return stdDev;
176     }
177 
178 
179     /* Getters */
180     public double getEdt() { return edt; }
181     public double getEt() { return et; }
182     public double getDt() { return dt; }
183     public double getEit() { return eit; }
184     public double getDit() { return dit; }
185 
186     /** We're lazy so functions for q10
187      * Col1 = D(t=end)
188      * Col2 = E(t=end) / D(t=end)
189      * Col3 = ED(t=end) / D(t=end)
190      * **/
191     public double col1() { return getDt(); }
192     public double col2() { return (getEt() / getDt()); }
193     public double col3() { return (getEdt() / getDt()); }
194 
195     public void printCols() {
196         String s = String.format("%.2f;%.2f;%.2f", col1(), col2(), col3());
197         System.out.println(s);
198     }
199 
200     public void printState() {
201         String ddt = "[";
202         String det = "[";
203         String dedt = "[";
204 
205         for (Double d : d_dt)
206             ddt += String.format(" %.2f\t", d);
207 
208         for (Double d : d_et)
```

```
209             det += String.format(" %.2f\t", d);
210
211         for (Double d : d_edt)
212             dedt += String.format(" %.2f\t", d);
213
214         ddt += "]";
215         det += "]";
216         dedt += "]";
217
218         String s = String.format("dit: %.2f\teit: %.2f\tdt: %.2f\tet: %.2f\tedt: %.2f\n"
219                         "d_dt:\t%s\n" +
220                         "d_et:\t%s\n" +
221                         "d_edt:\t%s\n",
222                 dit, eit, dt, et, edt, ddt, det, dedt);
223
224
225         System.out.println(s);
226     }
227 }
```