Devoir ARA 2017–2018

Jonathan Lejeune

Objectifs

L'objectif principal de ce projet est d'étudier les performances de différents algorithmes de diffusion de messages sur des réseaux mobiles ad hoc, appelés plus couramment MANET (Mobile Ad hoc NETworks).

Consignes générales

- Ce travail est à faire seul ou (de préférence) en binôme. Les trinômes ne sont pas autorisés.
- Les livrables sont à rendre par email à : « jonathan.lejeune@lip6.fr » le 4 février 2018, 23h59 au plus tard
- L'objet du mail doit être de la forme : [ARA-devoir2018] nom1-nom2
- Le mail contiendra 2 pièces jointes :
  - ◇ une archive des sources java (éviter les archives trop volumineuses), accompagnées d'un fichier texte "Readme" indiquant comment compiler le projet et lancer les différentes simulations (votre projet doit pouvoir se compiler/lancer en dehors d'Eclipse, l'utilisation d'un Makefile est vivement conseillée) ;
  - ◇ Votre rapport, au format pdf, concis, dans lequel vous devez répondre aux questions posées dans le sujet
- Il vous est interdit de modifier les interfaces Java qui vous ont été fournies mais rien ne vous empêche de les étendre.

Le non respect de ces consignes entraînera un malus sur la note globale du devoir.

Introduction

Un réseau ad-hoc est composé de nœuds possédant chacun un émetteur radio sans fil. Les communications ne se basent sur aucune infrastructure fixe et il n'y a aucune entité centrale responsable du routage des messages. En effet, les nœud peuvent communiquer directement avec leurs voisins, c'est à dire les nœuds présents dans leur rayon d'émission. Deux nœuds non voisins peuvent donc communiquer seulement si d'autres nœuds situés entre eux peuvent relayer les messages de hop en hop. De plus, les nœuds ont des contraintes énergétiques car ils se basent sur une batterie qui peut se décharger plus ou moins vite si l'émetteur radio est trop souvent sollicité. Dans les MANET, les nœuds

sont mobiles. Les liens réseaux entre voisins sont dynamiques et peuvent donc apparaître ou disparaître au cours du temps en fonction du mouvement de chaque nœud. Aucune garantie ne peut donc être assurée sur une connectivité de bout en bout.

La diffusion de messages dans un tel réseau reste une opération très courante car elle permet par exemple de propager une information sur tout le réseau ou bien de calculer et réserver une route entre deux nœuds. Cette opération doit :

- assurer que tout nœud du réseau accessible (à termes) depuis la source d'émission, reçoive le message
- minimiser les retransmissions, c'est à dire faire en sorte dans l'idéal qu'un nœud reçoive exactement une seule fois le message afin d'éviter de solliciter inutilement les batteries.

Préparation du projet

Créez un nouveau projet Eclipse, configurez le build path pour prendre en compte les jar de peersim (voir TP1) et y copiez les sources disponibles à l'URL suivante :

`https://pages.lip6.fr/Jonathan.Lejeune/documents/enseignements/ARA/ressources_devoir_ara_2017_2018.zip`

Assurez vous que la compilation se passe bien (absence de croix rouges). Dans les sources vous trouverez les packages suivants :

- `manet` : package contenant des classes ou interfaces utilitaires :
  - ◇ la classe `GraphicalMonitor` : control peersim permettant d"avoir un aperçu graphique de l'état du système. Ceci peut vous servir pour les phases de debug car il permet de mettre en pause, accélérer, ralentir la simulation.
  - ◇ la classe `MANETGraph` : permet de représenter si besoin le système sous forme de graph Peersim permettant ensuite d'y appliquer des algorithmes de graphes (par exemple : la connexité). Vous pouvez consulter l'interface `peersim.graph.Graph` et la classe `peersim.graph.GraphAlgorithms` pour plus d'informations.
  - ◇ la classe `Message` (voir TPs).
  - ◇ l'interface `Monitorable` : Utilisée par le contrôleur graphique, elle permet de récupérer des informations sur l'état du nœud notamment sur la partie applicative.
- `manet.algorihm.gossip` : composé d'une interface permettant de déclencher une diffusion. C'est cette interface que vous implémenterez pour coder vos algorithmes de diffusion.
- `manet.communication` composé de l'interface `Emitter` qui est un protocole permettant de simuler l'émission de messages dans un rayon géographique donné.
- `manet.detection` composé de deux interfaces :
  - ◇ `NeighborProtocol` : représente un protocole permettant de détecter les voisins directs d'un nœud
  - ◇ `NeighborhoodListener` : représente un protocole (applicatif) sensible aux changements du voisinage du nœud
- `manet.positioning` est composé de classes permettant de simuler le mouvement des nœuds sur un terrain.
- `manet.positioning.strategies` est composé de classes décrivant différentes stratégies de mouvement des nœuds.

Les unités considérées ici sont :

- des millisecondes pour les temps
- des mètres pour les distances
- des mètres par seconde pour les vitesses.

Exercice 1 – Implémentation d'un MANET dans PeerSim

Nous allons construire incrémentalement un modèle de MANET dans PeerSim. En premier lieu vous testerez le code fourni sur le mouvement et le positionnement des nœuds, puis vous coderez un protocole permettant l'émission de messages dans la portée radio d'un nœud et enfin un protocole permettant de détecter les nœuds voisins directs.

Question 1

En analysant le code de la classe `PositionProtocolImpl`, donnez l'algorithme général de déplacement d'un nœud. Il ne vous est pas demandé de code dans cette question.

Pour coder une politique de déplacement il faut définir deux types de stratégie :
- une stratégie de placement initial (SPI) sur le terrain
- une stratégie de déplacement (SD) en choisissant une nouvelle destination

Les classes du package `manet.positioning.strategies` peuvent décrire seulement une SPI (cas des stratégies 5 et 6), ou bien décrire seulement une SD (cas des stratégies 2 et 4) ou bien peuvent décrire les deux (cas des stratégies 1 et 3) si la SPI et la SD sont les mêmes.

Question 2

Testez le simulateur en prenant la stratégie 1 comme SPI et SD. Le contrôleur graphique sera déclenché toutes les unités de temps, son time_slow pourra être environ de 0.0002. Le seul protocole à renseigner pour ce contrôleur est le PositionProtocol de la simulation, les autres sont pour l'instant optionnels et sans objet. Normalement vous devez voir graphiquement des points verts se déplacer sur l'écran. Vous répondrez à cette question en donnant le contenu de votre fichier de configuration.

Question 3

Que fait la stratégie 1 ?

Question 4

Re-testez en prenant en SD, la stratégie 2 (la stratégie 1 reste la SPI). Que fait la stratégie 2 ?

Nous allons ajouter à présent le protocole permettant de simuler l'envoi de messages aux nœuds voisins

Question 5

Codez une classe implémentant l'interface `Emitter`. À l'émission d'un message (méthode `emit`) vous ajouterez un événement de réception dans la file d'événements du simulateur. Cet événement doit être délivré à tout nœud présent dans la portée de l'émetteur. La réception du message sera gérée par le protocole auquel le message est rattaché. Ça sera donc à ce protocole de filtrer en fonction du (ou des) destinataire(s) du message si il faut le délivrer ou pas. Testez de nouveau avec le moniteur graphique et assurez-vous que les portées sont représentées (cercle en bleu). Vous répondrez à cette question en donnant le code de votre classe.

Nous allons à présent, coder un protocole permettant de détecter les voisins. Ce protocole se basera sur un `Emitter` et implantera la classe `NeigborProtocol` qui impose de maintenir une liste de voisins. Pour détecter la présence des voisins on se basera sur un mécanisme de heartbeat. Les nœuds envoient périodiquement un message `Probe` dans leur champs d'émission. Tout nœud présent dans le champs d'émission qui reçoit pour la première fois le probe, ajoutera dans la liste, le voisin en question. Dans tous les cas, à chaque réception de probe, le nœud armera un timer pour chaque voisin connu. Si le timer se déclenche, un nœud peut alors considérer que le nœud associé n'est plus son voisin et le supprimera donc de sa liste.

Question 6

Codez une classe implantant l'interface `NeigborProtocol`. Votre classe devra prendre les paramètres de simulation suivants :

- la période d'envoi de message probe
- le temps de déclenchement du timer
- et optionnellement un protocole `NeighborhoodListener`. Ainsi, si ce paramètre est spécifié, la méthode `newNeighborDetected` doit être appelée lors de l'ajout d'un nouveau voisin et la méthode `lostNeighborDetected` doit être appelée lors de la perte d'un voisin.

NB : N'oubliez pas que vous êtes en simulation en temps discrétisé. L'utilisation de Timer Java out tout autre API utilisant le temps système ne peut pas fonctionner.

Question 7

Testez votre code, et remarquez sur le moniteur graphique que lorsque deux nœuds sont atteignables entre eux alors un lien graphique apparaît.

Question 8

Re-exécutez votre code avec les SPI et SD non encore testées et expliquez ce que fait chaque stratégie en précisant si elles assurent un graphe connexe à termes.

Par la suite, nous allons avoir besoin de caractériser la réseau par une densité, c'est à dire un nombre moyen de nœuds dans une surface donnée. Dans notre cas, nous allons définir la densité comme le nombre moyen de voisins par nœud.

Question 9

Codez un contrôleur `DensityControler` qui permet de calculer au moment de son déclenchement à l'instant t :

- $D_i(t)$: la moyenne du nombre de voisins par nœud à l'instant t.
- $E_i(t)$: l'écart-type de $D_i(t)$ permettant de juger si la valeur calculé en $D_i(t)$ est relativement homogène sur l'ensemble des nœuds. Plus cette valeur est élevée par rapport à la moyenne, plus la disparité de densité est importante, ce qui caractériserait un réseau où il existe des zones avec des voisinages très denses et des zones avec des voisinages peu denses.
- $D(t)$: la moyenne de l'ensemble des valeurs $D_i(t')$ pour tout $t' \leq t$, ce qui donnera une densité moyenne sur le temps.
- $E(t)$: la moyenne de l'ensemble des valeurs $E_i(t')$ pour tout $t' \leq t$, ce qui donnera la disparité moyenne de densité sur le temps.
- $ED(t)$: l'écart type des valeurs $D_i(t')$ pour tout $t' \leq t$, ce qui permettra de juger de la variation de la densité au cours du temps. Plus cette valeur est élevée par rapport à $D(t)$, plus le réseau a changé de densité moyenne au cours du temps.

Question 10

Complétez le tableau suivant pour les paramètres suivants :

- nombre de nœuds : 50
- Temps de simulation : 8 heures
- vitesse des noeuds : entre 1 et 20 m/s
- temps de pause : 20 minutes
- taille de terrain : 1500x1500 m
- latence d'émission : 90 msec
- période de probe : 3 secondes
- timer de probe : 3,5 secondes
- départ du `DensityControler` à 1 heure
- déclenchement du `DensityControler` toutes les 2 minutes

Vous pouvez désactiver le contrôleur graphique pour éviter de ralentir la simulation. Vous arrondirez au centième les valeurs calculées. Dans les deux dernières colonnes le fait de diviser $E(t_{end})$ et $ED(t_{end})$ par $D(t_{end})$ permet d'avoir un pourcentage, ce qui permet de comparer plus facilement les différentes stratégies.

portée	SPI	SD	$D(t_{end})$	$\frac{E(t_{end})}{D(t_{end})}$	$\frac{ED(t_{end})}{D(t_{end})}$
125	1	1			
250	1	1			
375	1	1			
500	1	1			
625	1	1			
750	1	1			
875	1	1			
1000	1	1			
125	3	3			
250	3	3			
375	3	3			
500	3	3			
625	3	3			
750	3	3			
875	3	3			
1000	3	3			

Question 11

Pour les deux stratégies considérées, quelle est l'impact de la portée sur la densité ?

Exercice 2 – Étude de protocoles de diffusion

Par la suite nous allons étudier les performances de différents protocoles de diffusion en fonction de plusieurs densités de réseau. Dans cet exercice nous avons besoins de considérer un graphe connexe à termes. Nous utiliserons donc comme SPI et SD, les stratégies 5 et 4 respectivement. Pour une diffusion, nous allons considérer les métriques suivantes :

- l'**atteignabilité** noté Att : le pourcentage de nœuds atteignables ayant reçu le message.
- l'**économie de rediffusion** noté ER qui est le pourcentage de nœuds qui ont reçu le message et qui n'ont pas rediffuser le message. Cette métrique peut se calculer par la formule suivante $\frac{(r-t)}{r}$ où r est le nombre de nœuds ayant reçu le message et t le nombre de nœuds qui ont retransmis le messages

Question 1

Nous avons besoin de calibrer dans un premier temps la configuration du réseau pour

connaître une densité moyenne donnée $D(t_{end})$ sur le temps. Pour que cette densité soit la plus représentative possible de la configuration, il faut que $\frac{ED(t_{end})}{D(t_{end})}$ soit le plus bas possible. Dans cette expérience, nous allons faire varier la densité en faisant varier le nombre de nœuds mais tout en fixant la portée et les paramètres $minDist$ et $maxDist$ des stratégies. Pour les paramètres suivants :

- portée des nœuds : 300 m
- `distance_init_min`, `distance_init_max`, `distance_min`, `distance_max` tous égaux à 200 m
- Temps de simulation : 12 heures
- vitesse des noeuds : entre 1 et 20 m/s
- temps de pause : 20 minutes
- taille de terrain : 1500x1500 m
- latence d'émission : 90 msec
- période de probe : 3 secondes
- timer de probe : 3,5 secondes
- départ du `DensityControler` à 1 heure
- déclenchement du `DensityControler` toutes les 2 minutes

Vous pouvez désactiver le contrôleur graphique pour éviter de ralentir la simulation. Vous arrondirez au centième les valeurs calculées.

Complétez le tableau ci-contre et tracez une courbe avec en abscisse la taille du réseau et en ordonnée $D(t_{end})$ afin que cela soit plus lisible et une autre courbe avec en ordonnée $\frac{ED(t_{end})}{D(t_{end})}$. Analysez l'évolution des courbes en fonction de la taille du réseau.

Rappel : analyser une courbe = expliquer le pourquoi de la forme d'une courbe.

taille réseau	$D(t_{end})$	$\frac{ED(t_{end})}{D(t_{end})}$
20		
30		
40		
50		
60		
70		
80		
90		
100		
120		
140		
160		
180		
200		

Par la suite vous prendrez les mêmes valeurs des paramètres listés en question 1. Notre protocole expérimental serait de lancer N diffusions séquentielles (utilisation de l'interface GossipProtocol) en choisissant à chaque diffusion, un nœud source aléatoirement. On doit assurer avant de lancer une diffusion i que la diffusion $i-1$ soit bien terminée. Attention, comme nous étudierons des algorithmes qui n'ont pas forcément une Att de 100 %, le fait de compter le nombre de nœuds ayant reçu le message n'est pas suffisant pour détecter la terminaison d'une diffusion.

Question 2

Expliquez votre démarche pour régler ce problème. Votre solution devra se faire de manière non intrusive, ni dans le code applicatif, ni dans le code qui vous a été fourni.

Question 3

Codez le contrôleur peersim `GossipControler` qui permet de :

- lancer les N diffusions séquentielles
- calculer pour chaque diffusion Att et ER.
- calculer la moyenne des Att et des ER (notés respectivement Att_N et ER_N) avec leur écart-type, une fois que les N diffusions sont terminées. Ce sont ces deux valeurs qui permettront d'évaluer les performances d'un algorithme.

Question 4

Le premier algorithme à tester est un algorithme de flooding simple. Le principe est que lorsqu'un nœud reçoit un message pour la première fois, il retransmet le message dans son rayon d'émission. Pour cette expérience, on prendra $N = 500$. Il vous est possible de désactiver la détection des voisins car cette information est inutile ici.

- Vous tracerez :
  - ◇ une courbe montrant l'évolution de Att_{500} en fonction de la densité
  - ◇ une courbe montrant l'évolution de ER_{500} en fonction de la densité
- Vous renseignerez pour chaque point calculé l'écart-type de Att_{500} et ER_{500}.
- Vous analyserez vos résultats

Question 5

Le deuxième algorithme est une approche probabiliste. Ainsi lorsqu'un nœud reçoit un message pour la première fois, il retransmet le message dans son rayon d'émission avec une probabilité p. Le tirage aléatoire se fera bien entendu en utilisant le random du simulateur (`CommonState.r`). Pour cette expérience il vous est possible de désactiver la détection des voisins car cette information est inutile ici. Vous testerez cet algorithme pour plusieurs valeurs de densité et pour chaque densité les valeurs de p suivantes : 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. Pour cet algorithme, vous tracerez un graphique qui a pour abscisse la densité et comme ordonnée, la probabilité p. Sur ce graphique vous tracerez trois courbes :

- une courbe composée de points où le Att_{500} calculé est supérieur ou égal à 99%
- la même courbe mais où le Att_{500} calculé est supérieur ou égal à 90%
- la même courbe mais où le Att_{500} calculé est supérieur ou égal à 80%

Ces courbes vous permettent donc de déterminer pour un seuil de Att_N donné, la probabilité p et la densité nécessaires pour atteindre ce seuil. Que remarquez vous ?

Question 6

Le troisième algorithme consiste à prendre en compte la densité locale c'est à dire la taille du voisinage V dans le calcul de la probabilité p. Rappelons nous que nous devons à la fois maximiser Att_N et ER_N. D'après les résultats de la question précédente, vaut-il mieux que la probabilité p soit proportionnelle (c'est à dire $p = kV$) ou bien inversement proportionnelle (c'est à dire $p = \frac{k}{V}$) à la taille du voisinage V ?

- Justifiez votre réponse
- Tracez pour plusieurs valeurs de k :
  - ◇ une courbe montrant l'évolution de Att_{500} en fonction de la densité
  - ◇ une courbe montrant l'évolution de ER_{500} en fonction de la densité
- Vous renseignerez pour chaque point calculé l'écart-type de Att_{500} et ER_{500}.
- Analysez vos résultats

Les deux algorithmes précédents ont l'inconvénient de ne prendre en compte qu'une constante ou bien qu'une vision locale pour le tirage aléatoire. Il serait cependant intéressant que le tirage se fasse en fonction de la position des autres voisins. On peut remarquer que plus deux nœuds voisins A et B sont proches plus la zone couverte par les deux zones d'émission est restreinte. Dans ce cas, si A envoie un message à B, il est donc peu utile que B retransmette le message car la plupart des voisins de B l'ont déjà reçu par A. À l'inverse si B se trouve à la limite du rayon d'émission de A, il y a des chances que beaucoup de voisins de B n'aient pas encore reçu le message

Question 7

Le quatrième algorithme consiste à calculer la probabilité de retransmission en fonction de la distance émetteur-récepteur. Ainsi $p = \frac{distance_{emetteur-recepteur}}{portee}$.

- Implémentez cet algorithme.
- Tracez :
  - ◇ une courbe montrant l'évolution de Att_{500} en fonction de la densité
  - ◇ une courbe montrant l'évolution de ER_{500} en fonction de la densité
- Vous renseignerez pour chaque point calculé l'écart-type de Att_{500} et ER_{500}.
- Analysez vos résultats

Les approches probabilistes précédentes ont le principal inconvénient que certains nœuds atteignables peuvent ne jamais recevoir le message si on a la malchance de choisir des mauvais nombres aléatoires. On propose donc de greffer un mécanisme permettant aux nœuds qui ont choisi de ne pas retransmettre, de vérifier tout de même si leurs voisins ont bien reçu le message. Pour ce faire, chaque nœud maintient pour chaque diffusion x une liste l_x de voisins qui n'ont potentiellement pas reçu le message. Les émetteurs doivent renseigner dans les messages la liste de leurs voisins notée V_{emet}. À la première réception d'un message d'une diffusion x, si le récepteur choisit de ne pas retransmettre en fonction de p alors :

- l_x est initialisée à la liste des voisins du récepteur ôtée de V_{emet} (tous les nœuds dans V_{emet} ont forcément reçu le message)
- un timer t_x est armé à une valeur aléatoire comprise entre T_{min} et T_{max}

Tant que le timer n'a pas expiré, à chaque réception d'un message m concernant la diffusion x, on ôte de l_x le V_{emet} renseigné dans m. À l'expiration d'un timer t_x, si l_x est non vide alors le message de la diffusion x est retransmis.

Question 8

Appliquez ce mécanisme sur les algorithmes 3 et 4 et refaites vos expériences. Quel est l'impact de ce mécanisme sur les métriques Att et ER ?

Question 9

Synthétisez l'intégralité des résultats obtenus dans cet exercice en précisant entre autre quel algorithme il est préférable de choisir pour une densité donnée.