Monitorage et journalisation dynamiques des topologies dans les réseaux ad-hoc

Cristian Popi, Olivier Festor

Monitorage et journalisation dynamiques des topologies dans les réseaux ad-hoc

Cristian Popi — Olivier Festor

MADYNES - INRIA Nancy Grand Est - Research Center[1]
615, rue du jardin botanique
54602 Villers-les-Nancy, France
{cristian.popi,festor}@loria.fr

RÉSUMÉ. *Nous étudions l'utilisation d'un service pair-à-pair basé sur des DHT (tables de hachage distribuées) pour le monitorage et la journalisation distribuée de l'historique de topologies sur un réseau ad-hoc. Notre approche tolère le fusionnement et le partitionement temporel d'overlay de monitorage. Elle permet ainsi à l'ensemble de fonctions de gestion (par exemple fautes, performance, configuration) de reconstruire un modèle de la topologie de réseau à tout instant de son historique. Nous présentons le protocole et le modèle de slot de temps utilisés par notre approche. L'architecture du système est décrite et son comportement est évalué par simulation.*

ABSTRACT. *We investigate the use of a Distributed Hash Table based Peer-to-Peer service to maintain the distributed monitoring history of an ad-hoc network topology. Our approach supports merging and splitting of monitoring overlays over time and enables various management functions (e.g. fault, performance, configuration) to rebuild a model of the network topology at any time in the past of its history. The protocol and time-slot models used are presented. The architecture of the supporting system is described and its performance is evaluated by simulation.*

MOTS-CLÉS : *réseau ad-hoc, monitorage, overlay*

KEYWORDS: *ad-hoc networks, monitoring, overlay*

1. Ce travail a été soutenu par le projet RNRT Airnet

1. Introduction

Un réseau ad-hoc est un réseau sans fil multi-sauts, dont les nœuds participants coopèrent pour acheminer des données ou découvrir des services. Un tel réseau est décentralisé et ne comporte pas nécessairement d'infrastructure préétablie. Les nœuds sont mobiles, et les liens entre eux instables.

La connaissance temporelle de la topologie d'un réseau ad-hoc est utile pour des nombreuses fonctions de gestion. La gestion des erreurs, par exemple, benefice de la disponibilité d'une telle information, puisque la détection des fautes dans le système est basée le plus souvent sur l'analyse des événements passés [QIU 06]. Dans les réseaux sans fil mobiles multi-radio, utiliser un historique de la topologie peut mener à diverses stratégies pour améliorer la QoS, en créant par exemple des algorithmes qui assignent des canaux aux interfaces radios d'un nœud [MAR 05]. Par conséquent, nous soulignons la nécessité et l'importance de maintenir un historique de la topologie du réseau dans un environnement dynamique et incertain tel que celui d'un réseau ad-hoc.

Nous proposons une approche distribuée d'observation de la topologie et supportant la journalisation de l'évolution de celle-ci sur une période de temps donnée au sein du réseau ad-hoc. Les nœuds participants rassemblent l'information de voisinage pendant de courts intervalles de temps, et la stockent aléatoirement à travers les autres nœuds dans le réseau. De plus, les nœuds ad-hoc se doivent d'offrir une interface pour la requête de topologie. Un gestionnaire peut alors interroger n'importe quel nœud du réseau pour obtenir les données de voisinage concernant un nombre défini d'intervalles de temps et ainsi reconstruire des images de la topologie physique.

Offrir un tel service sur un réseau ad-hoc est un véritable défi vue la mobilité des nœuds et les fluctuations des liens qui engendrent des changements fréquents de la topologie et donc un surcoût de messages et de traitement pour fournir en permanence une vue la plus complète et la plus juste possible de celle-ci à des applications de supervision. Afin que le plan de supervision soit adapté aux conditions spécifiques des réseau ad-hoc, celui-ci doit intégrer la nature dynamique de la topologie supervisée et doit impérativement être conscient de l'incertitude liée aux données qu'il va pouvoir collecter pour la représenter à un moment donné. De plus, le coût du plan de collecte d'information de gestion doit être maîtrisé ; un idéal inatteignable serait un coût nul. Notre approche est entièrement distribuée sur les nœuds participant au réseau ad-hoc. L'intergiciel de maintenance des données collectées s'appuie sur une architecture pair-à-pair structurée, maintenant les informations de topologies collectées dans le temps dans une table de hachage distribuée. Cette approche est intéressante pour (1) offrir un service d'accès à l'information de gestion à tout nœud participant qu'il soit gestionnaire ou agent, (2) pour distribuer la charge de stockage dans le temps sur le plus grand nombre de nœuds et de façon la plus équilibrée possible. Nous avons retenu Chord [STO 01] comme modèle pair-à-pair d'accueil, étendu avec un support de réplication des informations et un service de fusion/fission de tables de hachage. Nous y avons ajouté un algorithme de stockage des données par période temporelle. L'ar-

chitecture et les algorithmes spécifiques au traitement de données topologiques sont décrits dans cet article.

Le papier est structuré comme suit : la section 2 présente les travaux relatifs ainsi que des solutions sur lesquelles notre travail est construit. La section 3 présente l'architecture du système. La représentation des données topologiques et leur collecte sont décrites dans la section 4. La section 5 détaille le protocole de coopération entre les nœuds du système pour établir et maintenir l'information de topologie recueillie. Ensuite, l'algorithme pour construire la matrice de topologie à partir de l'information de topologie est détaillé (section 6). L'évaluation du framework proposé est réalisée dans la section 7. Celle-ci est suivie d'une conclusion.

2. Travaux Relatifs

DAMON [RAM 04] propose une architecture de surveillance distribuée pour les réseaux ad-hoc, basée sur l'existence de centres de dépôt, appelés puits, dans le réseau. Une organisation distribuée des puits est privilégiée par rapport à une organisation centralisée pour des raisons de passage à l'échelle. Les nœuds candidats pour jouer le rôle de puit sont des nœuds avec une mobilité réduite. Ceci fait des points d'accès, lorsqu'ils existent, des choix parfaits pour surveiller les nœuds clients mobiles qui leurs sont attachés, mais constituent des points de faille. Nous fournissons une approche entièrement distribuée ne présentant pas la vulnérabilité d'un unique point de collecte.

OLSR [CLA 03], peut en soi fournir la topologie du réseau. L'état partiel de lien est inondé par des nœuds participants, de sorte que les chemins les plus courts peuvent être calculés. Si les nœuds annoncent tout le voisinage, alors tous les nœuds dans le réseau peuvent avoir une pleine connaissance de la topologie du réseau. Le problème du maintient de cette information dans le temps reste présent. L'utilisation d'une approche de DHT (table de hachage distribuée) pour distribuer uniformément les données rassemblées avec un degré minimum de réplication (pour prévoir le cas de perte des nœuds) est une approche possible pour cette fonction. En outre, on peut déployer des réseaux ad-hoc avec une multitude de protocoles de routage développés ces dernières années, tels AODV [PER 03] ou DSR [JOH 07]. Si ils sont employés pour le routage, tirer profit des informations de topologie fournies par OLSR ne sera plus possible. Le fait qu'il y ait des protocoles de routage multiples renforce le besoin d'une solution qui soit indépendante de la couche de routage.

Nous proposons l'utilisation d'un overlay basé sur Chord [STO 01]. Dans la littérature de nombreux travaux ont porté sur l'adaptation de Chord aux réseaux ad-hoc [LAN 06, CAE 06, CAS 06, DAB 01a, DAB 01b]. Deux modèles ont notamment été proposés par [CHE 06] pour la composition dynamique de DHT : un modèle d'absorption et un modele par gateway. Une approche similaire, mais plus appropriée pour fusionner les tables de deux sous-partitions a été présentée dans [HEE 06]. Nous utilisons cette dernière dans notre approche.

3. Architecture de la solution

Notre solution de gestion comporte trois couches (fig. 1). La couche *application de gestion* utilise les services offerts par la DHT sous-jacente pour stocker/chercher l'information de topologie. Chaque nœud a la responsabilité de rassembler l'information de voisinage (comme présenté dans la section 4). Nous divisons le temps en intervalles discrets, que nous appelons des *slots*. Pour un slot de temps, que nous choisissons d'une longueur parametrable pour saisir suffisamment d'information de voisinage, et pour permettre également à un manager d'observer la dynamique du réseau, nous inférons le voisinage d'un nœud, et créons un enregistrement topologique, en associant l'information de voisinage au slot courant. Les enregistrements topologiques obtenus par chaque nœud, et pour chaque slot sont stockés dans la DHT recouvrant l'ensemble des nœuds contrôlés.

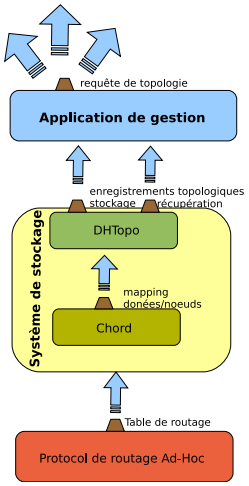

Figure 1. *Architecture du system de gestion avec les services offerts par chaque couche*

Le système de stockage est représenté dans la figure 1 par les couches *Chord* et *DHTopo*. *Chord* est responsable d'associer des identifiants aux nœuds, de construire et de maintenir les tables de routage overlay, ainsi que de la recherche des nœuds qui sont responsables d'un indentifiant donné. *DHTopo* emploie alors le service de recherche offert par *Chord* pour trouver le nœud sur lequel un enregistrement topologique est stocké. Le rôle de *DHTopo* est de stocker correctement les enregistrements topologiques concernant le même intervalle de temps pour différents nœuds. Une opération de *merge* est définie pour stocker les enregistrements des nœuds multiples sous la même clef Chord, pour le slot de temps courant, t_i. Le système de stockage fournit les services suivants à l'application de gestion : stockage sous la même clef des enregistrements de topologie (pour les mêmes intervalles de temps), et la récupération des enregistrements topologiques en une seule opération pour un slot de temps. La couche

application de gestion contrôle les slots de temps, organise l'information de voisinage et ajoute l'enregistrement ainsi obtenu dans la DHT en utilisant le service offert par la couche DHTopo. D'autre part, elle offre un service de requête aux gestionnaires souhaitant reconstruire un modèle de la topologie du réseau à un moment donné. Une requête peut être effectuée sur n'importe quel nœud de la communauté de gestion.

4. Collection et représentation des données

Puisque notre intérêt principal est l'information de la topologie, une représentation correcte devrait au minimum se composer d'une identification de chaque nœud, id, et d'un lien entre chaque paire de nœuds, $\mathcal{F}$, tels que, pour l'ensemble de tous les nœuds contrôlés $\mathcal{N}$, $\mathcal{F} : \mathcal{N}X\mathcal{N} \rightarrow \{'one\_hop\_neighbor', 'not\_neighbor'\}$. $\mathcal{F}(x, y) =' one\_hop\_neighbor'$, si pour $x, y \in \mathcal{N}$, y est un voisin de x, c'est à dire, qu'il y a un lien unidirectionnel entre x et y, et $\mathcal{F}(x, y) =' not\_neighbor'$ s'il n'y a aucun lien direct entre x et y.

Pour l'identification d'un nœud, nous avons retenu l'adresse MAC des interfaces sans fil de chaque nœud. Chaque nœud exécute un agent responsable d'intercepter des *beacons* venant des nœuds voisins. Nous organisons alors le voisinage de n'importe quel nœud n_i, comme une liste d'adresses MAC, $n_i \rightarrow \{n_{i1}, n_{i2}, ..., n_{il_i}\}, \forall n_i \in \mathcal{N}$, où l_i définit le nombre de voisins d'un nœud n_i.

Slots de temps

Nous divisons le temps en slots, t_i, de durée parametrable (dans nos simulation cet intervalle est fixé à 5 secondes). Chaque slot commence à un moment en temps (absolu) multiple de l'intervalle (e.g. 2008 January 13, 22 :24 :05). La topologie est capturée pour chaque slot. Pour tous les slots de temps t_i, les actions que l'agent effectue sur n'importe quel nœud n_k, sont les suivantes. Au début du slot de temps, il initialise sa liste de voisins ; le premier élément présent dans cette liste pour chaque slot est l'adresse MAC du nœud lui-même. Pour la durée entière du slot de temps, l'agent intercepte des beacons, les analyse, et si leur nombre est au-dessus d'un seuil défini, il ajoute l'adresse MAC du nœud envoyant les beacons dans la liste de voisinage ; si l'adresse est déjà dans la liste, toute autre beacon venant de cette adresse pour ce slot de temps est négligé. À la fin du slot l'agent crée l'enregistrement topologique, en datant la liste des voisins obtenue avec le slot de temps afférent, et l'ajoute à la DHT après attente d'une période de temps aléatoire (pour éviter de surcharger les nœuds responsables de l'hébergement des enregistrements topologiques pour ce slot de temps).

Chaque nœud, à la fin d'un slot de temps, appelle la primitive **put**(*timestamp*, *list*), fourni dans l'API de DHTopo. Puisque nous choisissons des slots de temps comme clefs en raison de leur unicité, les données de voisinage rassemblées depuis différents nœuds pour un certain slot de temps t_i seront mappées vers le même identifiant, et

stockées ainsi sur le même nœud de la DHT. DHTopo veillera à ce que les données venant de différents nœuds pour le même slot de temps ne soient pas écrasées (comme cela se produirait avec un fonctionnement standard de Chord), mais fusionnées dans une liste :

$$\{t_i, \{n_1, n_{11}, n_{12}, ..., n_{1l_1}\}, \{n_2, n_{21}, n_{22}, ..., n_{2l_2}\}, ...\} \quad [1]$$

et stockées sur le nœud dont l'identifiant est le plus petit identifiant supérieur a la clef obtenue à partir du hachage de t_i, et sur un certain nombre, r, de successeurs de la liste de successeurs du nœud de taille q, où $r \leq q$, comme représenté sur la figure 2.

Nous définissons un nombre maximum de slots de temps, T_N, qui peuvent être maintenus par le système. Nous contrôlons les T_N slots de temps de la façon First In First Out (FIFO), telles que les données stockées les plus anciennes seront toujours les premières à être remplacées par des données fraîchement rassemblées de voisinage. Pour une liste ordonnée de slots de temps t_0, t_1..., t_{T_N}, l'agent fonctionnant sur un nœud supervisé, a, fera ce qui suit :

– si le slot de temps courant, t_k, est inférieur à t_{T_N}, alors il pousse le tuple $\{t_k$, {a, neighborhood(a)}$\}$ dans la DHT et il maintient un $latest\_slot$ égal à t_k ; ceci permettra de rejouer d'une manière ordonnée des données topologiques (par exemple $(dernier\_slot, t_{T_N})$ suivi de $[t_0, dernier\_slot]$) ;

– si le slot de temps courant, t_k, est t_{T_N}, alors il met le slot courant sur t_0 et il pousse $\{t_0$, {a, neighborhood(a)}$\}$ dans la DHT ; pour éviter d'ajouter l'information fraîche à la même liste contenant de vieilles information (pour le même slot de temps), au début de t_{i-1}, chaque nœud vérifie s'il est responsable de l'enregistrement topologique lié au slot de temps t_i ; s'il l'est, alors il vide la liste de l'équation 1 de son contenu.

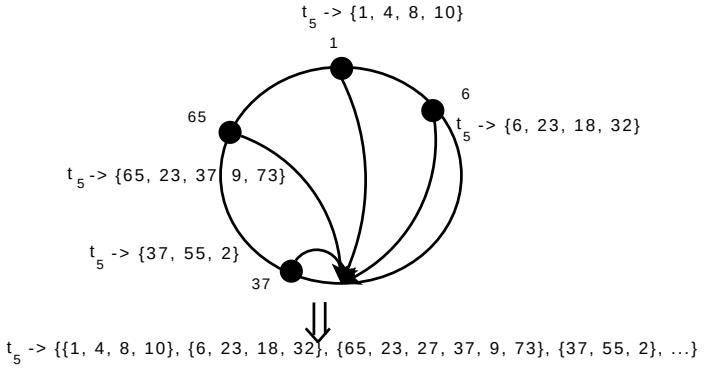

Figure 2. *Exemple de fusion des données de voisinage rassemblées dans une seule liste de type $\{n_k, n_{k1}, n_{k2}, ..., n_{kl_k}\}$, où l_k représente le nombre de voisins du nœud n_k ; le slot de temps t_5 se mappe au nœud ayant l'identifiant id=37 ; alors toute donnée de voisinage des nœuds 1, 6, 37, 65,... pour le slot t_5 sera stockée sur le nœud* 37 *dans une liste formée par la fusion des listes de voisinage individuelles*

5. Initialisation et maintenance de groupes supervisés

Chaque nœud de la communauté de gestion envoie périodiquement (chaque seconde) un message en broadcast avec un TTL=1, annonçant l'existence d'une organisation de gestion. Un TTL de 1 est suffisant pour faire participer tous les nœuds voisins (à une distance d'un hop) de la DHT, puisqu'après s'avoir rejoint le plan de monitorage, un nœud se comporte d'une façon similaire, prolongeant de ce fait la couverture de la DHT incrémentalement. Ce qui suit décrit ce qui se passe quand un nœud aléatoire, ne participant pas dans la DHT, reçoit l'annonce de l'existence de la DHT. Un nœud extérieur au groupe contrôlé qui reçoit l'annonce, accepte de se joindre et envoie un message de confirmation pour prouver qu'il est prêt à participer au plan de monitorage. Le nœud qui a fait l'invitation lance alors la procédure de rattachement du nouvel arrivé dans la DHT ; il hache l'IP du nœud invité, puis cherche un successeur pour le nouveau nœud ; il envoie alors de nouveau au nœud invité le successeur dans l'anneau de Chord. Parce-que les nœuds sont a priori synchronisé, le nœud peut commencer à collecter des informations de voisinage pour chaque slot et à les mettre dans la DHT.

Joindre deux DHTs séparées avec la même fonction (pour stocker l'information topologique) n'affectera pas les enregistrements topologiques dans les deux groupes fusionants, pour le même slot de temps, parce que l'opération de *merge* définie dans la couche *DHTopo* fusionnera les deux listes de paires $node-neighbors$. Ce mécanisme permet à un gestionnaire de faire une seule requête sur un des nœuds contrôlés après la fusion des deux partitions, pour obtenir une vue unitaire de la topologie du réseau pour un slot de temps correspondant à un moment où le réseau était divisé.

La procédure de bootstrapping est dans ce cas la suivante. Tous les nœuds écoutent pour une période de temps aléatoire (d'un maximum défini que nous avons fixé à 10 secondes dans notre simulation) pour de broadcasts annonçant l'existence d'une organisation de gestion. Après cette temps d'attente, chaque nœud qui a pas reçu un broadcast (et donc qui n'a pas rejoint un plan de monitorage) va créer une DHT, et va commencer à envoyer des broadcasts annonçant l'existence de sa DHT. Le nœud ne dépose des enregistrements dans la DHT qu'à partir d'un multiple de la durée du slot (5 seconds). Ce mécanisme assure qu'il aura un moins un nœud parmi tous les nœuds qui va bootstrapper le processus de collection/stockage de données topologiques. Intégrer deux DHTs revient donc au cas de la fusion décrit dans le paragraphe précédent.

En ce qui concerne la synchronisation, on fait l'hypothèse que les nœuds sont a priori synchronisés.

6. Algorithme pour la construction de la matrice de topologie

Un gestionnaire qui veut consulter la topologie pour un intervalle donné de temps, de t_i à t_j, doit faire $j-i+1$ opérations de *lookup* pour pouvoir rejouer la topologie du réseau pour cet intervalle. L'algorithme pour établir la matrice de topologie est décrit ci-dessous.

Input: $t_i..t_j$
Output: Topology matrices $\mathcal{M}_k$ for each time slot t_k

```
1  for t_k = t_i to t_j do
2      NeighborhoodList = lookup(t_k);
3      M_L={};
4      foreach node s ∈ the NeighborhoodList do
5          if s ∉ M_L then
6              add s to M_L;
7              foreach p ∈ M_L do
8                  M_k(s,p) = M_k(p,s) = 0;
               end
           end
9          foreach r ∈ NeighborList_s do
10             if r ∉ M_L then
11                 add r to M_L;
12                 foreach p ∈ M_L do
13                     M_k(r,p) = M_k(p,r) = 0;
                   end
               end
14             M_k(r,s) = 1
           end
       end
   end
```

Algorithm 1: L'algorithme pour construire et rejouer les matrices de topologie associées à une période [t_i, t_j]

$\mathcal{M}_k$ représente la matrice de topologie pour le slot de temps t_k et $\mathcal{M}_\mathcal{L}$ dénote l'ensemble des éléments index de la matrice carrée $\mathcal{M}_k$. À la deuxième ligne de l'algorithme, la liste $NeighborhoodList$ (ayant la forme de l'équation 1) est peuplée, en consultant l'entrée de la clef qui est responsable du slot t_k, et puis en obtenant les données de voisinage du nœud hébergeant la clef (la fonction *lookup* dans l'algorithme comporte en réalité deux appels de fonction : un pour rechercher les données topologiques, et l'autre pour les retirer). La ligne 3 initialise l'ensemble des indexes de la matrice, $\mathcal{M}_\mathcal{L}$. Puis, la matrice est construite, en parcourant toutes les sous-listes de $NeighborhoodList$, appelés $NeighborList_s$, qui contiennent les voisins de nœud s. Nous ajoutons s et ses voisins, r, dans $\mathcal{M}_\mathcal{L}$, et nous mettons $\mathcal{M}_k$ (r, s) à "1", pour montrer qu'il y a un lien direct entre r et s.

7. Évaluation du système

Nous avons implanté le framework de supervision distribué dans le simulateur OmNet++ [VAR 01] afin d'étudier sa faisabilité et son comportement. La couche physique

est simulée avec le modèle de propagation de disque-unité, avec une portée de transmission de 100m, et la couche liaison de données est basée sur le modèle IEEE 802.11 MAC. Au niveau du routage nous employons le protocole AODV. Ils font tous partie du framework existant INET, qui a été modifié pour contenir le protocole AODV. Au-dessus d'INET existe un framework de simulation pair-à-pair, OverSim, qui simule le protocole Chord, et la DHT construite sur Chord.

Dans la première série d'expériences nous avons fait bouger 10 et respectivement 100 nœuds dans un carré de 1000m x 1000m avec une vitesse aléatoire uniformément distribuée entre 15 et 20 m/s, et avec un temps d'attente aléatoire uniformément distribué entre 3 et 8 secondes. Pour la deuxième série d'expériences nous avons changé la vitesse des nœuds entre 5 m/s et 10 m/s. Pour chacune des ces deux séries d'expériences, nous avons choisi au hasard un slot de temps, t_x, et nous avons alors mesuré à chacun des 50 slots de temps suivants, t_y, l'exactitude du système, en comparant la matrice obtenue après une requête de topologie pour le slot de temps t_x, $M_{Q_{xy}}$, à la vraie matrice de topologie, M_{R_x}, sur la période t_x. L'exactitude mesure le pourcentage des entrées correctes dans la matrice de topologie obtenue par interrogation.

Nous avons ensuite comparé la précision de notre proposition, à la précision d'un système centralisé, dont un nœud, N, agit comme un collecteur de données topologiques. Pour chaque slot de temps, tous les nœuds retiennent la liste de leurs voisins (comme décrit dans la section 4). Le nœud collecteur est choisi aléatoirement parmi les nœuds de l'expérience. Ce nœud envoie des requêtes de topologie à chaque nœud, i (qui s'annoncent au préalable) pour chaque slot de temps. Pour chaque requête N attend une réponse contenant la liste des voisins pour le slot contenu dans la requête. Les requêtes sont envoyées périodiquement. Pour optimiser le nombres de messages envoyés dans le réseau, le collecteur transmet à chaque nœud, i, dans une seule requête, tous les slots pour lesquels i n'a pas répondu. La précision du système de polling est calculée comme décrit ci-dessus, c'est à dire, en comptant le nombre des entrées correctes de la matrice obtenue par polling, par rapport à la matrice de topologie réelle M_{R_x} pour un slot t_x.

Résultats

Les résultats pour deux vitesses différentes de mouvement de nœuds (5-10 m/s et 15-20 m/s) sont présentés dans la figure 3(a,b,c,d). Les figures montrent l'exactitude moyenne du système de supervision pour un slot de temps choisi au hasard, t_x, en changeant le niveau de réplication de stockage dans la DHT, pour un nombre de nœuds entre 10 et 100. L'exactitude augmente avec le nombre de slots qui passent après le slot de temps t_x et se stabilise après une certaine période de temps écoulée entre le moment de la mesure et celui de la consultation. La période dont le système a besoin pour se stabiliser est le temps qu'il lui faut pour agréger complètement les enregistrements de topologie pour t_x se retrouvant dans les DHTs sur la plateforme expérimentale ; l'incomplétude topologique de l'information est due au manque de contact initial entre des nœuds ou des zones des nœuds. C'est pourquoi, il faut autour de 150 secondes

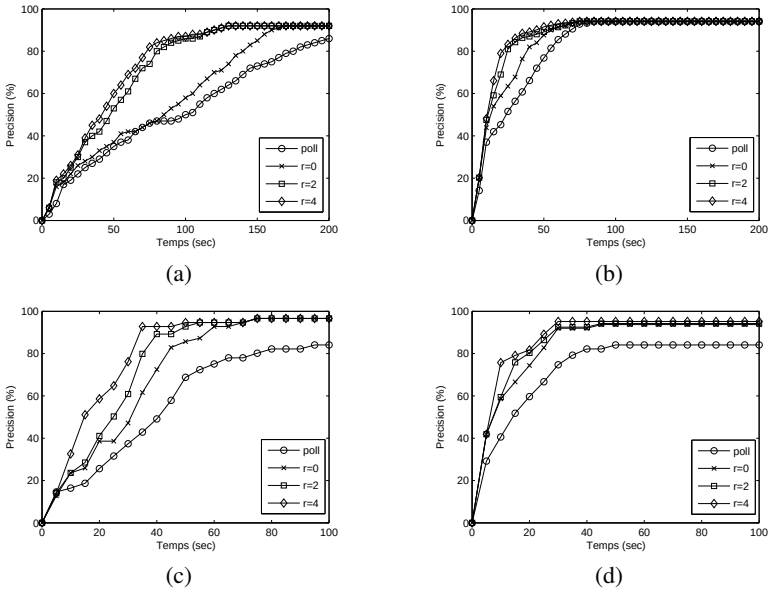

Figure 3. *Évaluation de l'exactitude moyenne du framework de supervision proposé vs. un système centralisé pour un nombre de 10 et 100 nœuds, qui bougent avec une vitesse aléatoire uniformément distribuée de 5-10 m/s et 15-20 m/s. Le niveau de réplication, r, de la DHT varie pour chacune des expériences. (a) 10 nœuds à 5-10 m/s ; (b) 100 nœuds à 5-10 m/s ; (c) 10 nœuds à 15-20 m/s ; (d) 100 nœuds à 15-20 m/s.*

(ou 30 slots de temps - fig. 3a) pour que l'exactitude du système soit élevée pour un nombre de 10 nœuds, mais elle prend seulement autour de 60 secondes (ou 12 slots de temps - fig. 3c) pour un nombre de 100 nœuds dans les mêmes conditions. Deux variables influencent l'exactitude du système : la vitesse avec laquelle les nœuds se déplacent et le niveau de la réplication des enregistrements topologiques dans la DHT. Une augmentation de chacun des ces deux paramètres diminue le nombre de slots de temps nécessaires pour collecter la topologie complète. Avec une vitesse accrue, les nœuds (ou les groupes isolés de nœuds) prennent moins de temps avant d'arriver à portée de transmission d'autres groupes de nœuds et fusionner leur DHTs ; cela améliore ainsi l'information topologique stockée.

En comparant les deux systèmes (dans la figure 3, *poll* représente la précision moyenne du système de polling), on conclut, qu'en moyenne, la précision du système DHT converge plus vite, vers une valeur élevée. Cela devient plus important avec l'augmentation du niveau de réplication dans la DHT. La raison est que dans le système de polling le collecteur doit "voir" tous les nœuds pour avoir l'ensemble de la topologie. Par contre, avec une DHT, il suffit d'avoir au moins deux nœuds qui détiennent des données topologiques complémentaires pour le slot t_x (e.g. de deux par-

titions différentes au moment t_x), pour pouvoir reconstruire toute la topologie. Plus la réplication augmente, plus il y a de chance que deux nœuds (ou groupes) se rencontrent et fusionnent leurs DHTs, contribuant ainsi à l'augmentation de la précision du système de monitorage.

Une exactitude parfaite est possible. Dans le simulateur cela signifie que la matrice de topologie réelle (celle donnée par le simulateur à la fin du slot de temps pour lequel nous interrogeons la topologie) est identique à celle obtenue par construction a partir de la liste de voisinage de chaque nœud dans la DHT. Pour être plus précis, les nœuds qui figurent dans la liste de voisinage du nœud N (ceux qui envoient un nombre de beacons supérieur au seuil défini) doivent toujours être voisins de N à la fin du slot de temps, quand le simulateur construit la matrice de topologie réelle.

8. Conclusion et perspectives

Nous avons proposé un framework de monitorage de topologie pour des réseaux ad-hoc, basé sur des tables de hachage distribuées. Le système stocke l'information topologique recueillie par les clients mobiles, à des intervalles de temps discrets, que nous avons appelés des slots de temps. Les instantanés de la topologie sont pris pour chaque slot de temps et sont stockés de manière distribuée, avec un équilibre équitable de charge sur chaque nœud dans le système. Le système peut alors être interrogé à tout moment pour connaître la topologie dans n'importe quel slot de temps de l'historique par un ou plusieurs gestionnaires auprès de n'importe quel nœud du réseau. Nous avons choisi de baser notre système de monitoring/journalisation sur un système pair-à-pair implantant une DHT avec réplication, car elle offre deux avantages importants : la résilience aux échecs de nœuds, et deuxièmement, le passage à l'échelle offert par Chord permet son application à un environmnet ad-hoc avec un nombre important de nœuds impliqués. Nous avons proposé une solution à la division provisoire du réseau. Celle-ci permet une construction de la topologie à partir de deux composants séparés, après leur fusion. Une fois que les deux sous-réseaux sont fusionnés, nous pouvons alors rejouer la topologie même pour la période de temps avant leur division.

En perspective, nous souhaitons étudier plus avant les influences des paramètres retenus pour notre étude (e.g. 5 s pour la durée d'un slot de temps), et faire de simulations à plus grande échelle sur plusieurs milliers de nœuds en analysant la surcharge due à notre approche. Ensuite, en s'appuyant sur notre système, nous projetons son utilisation pour collecter d'autres données telle la qualité des liens comme dans [KIM 06], et les quantifier sur une échelle (ex. de 1 à 5), pour pouvoir construire une matrice de topologie, qui au lieu de déclarer simplement qu'il y a un lien entre a et b, établit le niveau de fiabilité des liens.

9. Bibliographie

[CAE 06] CAESAR M., CASTRO M., NIGHTINGALE E. B., O'SHEA G., ROWSTRON A., « Virtual ring routing : network routing inspired by DHTs », *SIGCOMM '06*, ACM Press, 2006, p. 351–362.

[CAS 06] CASTRO M., O'SHEA G., ROWSTRON A., « Zero servers with zero broadcasts », *MobiShare '06*, ACM Press, 2006, p. 36–41.

[CHE 06] CHENG L., OCAMPO R., JEAN K., GALIS A., SIMON C., SZABO R., KERSCH P., GIAFFREDA R., « Towards Distributed Hash Tables (De)Composition in Ambient Networks. », *DSOM*, vol. 4269, Springer, 2006, p. 258-268.

[CLA 03] CLAUSEN T., JACQUET P., « Optimized Link State Routing Protocol (OLSR) », RFC Experimental n° 3626, Oct 2003, Internet Engineering Task Force.

[DAB 01a] DABEK F., BRUNSKILL E., KAASHOEK M. F., KARGER D., MORRIS R., STOICA I., BALAKRISHNAN H., « Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service », *HotOS-VIII*, Schloss Elmau, Germany, 2001, IEEE Computer Society.

[DAB 01b] DABEK F., KAASHOEK M. F., KARGER D., MORRIS R., STOICA I., « Wide-area cooperative storage with CFS », *SOSP '01*, ACM Press, 2001, p. 202–215.

[HEE 06] HEER T., GOTZ S., RIECHE S., WEHRLE K., « Adapting Distributed Hash Tables for Mobile Ad Hoc Networks », *PERCOMW '06*, IEEE Computer Society, 2006.

[JOH 07] JOHNSON D. B., MALTZ D. A., HU Y. C., « The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR) », rapport, February 2007, IETF MANET Working Group.

[KIM 06] KIM K.-H., SHIN K. G., « On accurate measurement of link quality in multi-hop wireless mesh networks », *MobiCom '06*, New York, NY, USA, 2006, ACM Press, p. 38–49.

[LAN 06] LANDSIEDEL O., GÖTZ S., WEHRLE K., « A churn and mobility resistant approach for DHTs », *MobiShare '06*, New York, NY, USA, 2006, ACM Press, p. 42–47.

[MAR 05] MARINA M., DAS S., « A topology control approach for utilizing multiple channels in multi-radio wireless mesh networks », *2nd International Conference on Broadband Networks*, vol. 1, Oct. 2005, p. 381–390.

[PER 03] PERKINS C. E., BELDING-ROYER E. M., DAS S. R., « Ad hoc On-Demand Distance Vector (AODV) Routing », RFC Experimental n° 3561, July 2003, Internet Engineering Task Force.

[QIU 06] QIU L., BAHL P., RAO A., ZHOU L., « Troubleshooting wireless mesh networks », *SIGCOMM Comput. Commun. Rev.*, vol. 36, n° 5, 2006, p. 17–28, ACM Press.

[RAM 04] RAMACHANDRAN K., BELDING-ROYER E., AIMEROTH K., « DAMON : a distributed architecture for monitoring multi-hop mobile networks », *IEEE SECON '04*, 2004, p. 601–609.

[STO 01] STOICA I., MORRIS R., KARGER D., KAASHOEK M. F., BALAKRISHNAN H., « Chord : A scalable peer-to-peer lookup service for internet applications », *SIGCOMM '01*, New York, NY, USA, 2001, ACM Press.

[VAR 01] VARGA A., « The OMNeT++ discrete event simulation system », *European Simulation Multiconference (ESM'2001)*, June 2001.