Conception et expérimentation d'un protocole de collecte de données pour réseaux de véhicules

Yoann Dieudonné* — **Bertrand Ducourthial*** — **Sidi-Mohammed Senouci****

Université de Technologie de Compiègne
Heudiasyc, UMR CNRS 6599
BP. 20529, 60205 COMPIEGNE Cedex, France

{yoann.dieudonne,bertrand.ducourthial}@hds.utc.fr

Université de Bourgogne (Orange Labs Lannion au moment de l'étude)
Institut Supérieur de l'Automobile et des Transports (ISAT)
49, rue Mademoiselle Bourgeois 58000 Nevers

Sidi-Mohammed.Senouci@u-bourgogne.fr

RÉSUMÉ. *Dans cet article, nous nous intéressons au problème de la collecte de données au sein d'un réseau ad hoc véhiculaire (en anglais, Vehicular Ad hoc Network ou VANET). À cet effet, nous proposons un protocole garantissant, de manière concomitante, trois propriétés pouvant s'avérer avantageuses dans un réseau intrinsèquement dynamique et donc en proie à d'incessants changements topologiques. La première de ces propriétés réside dans le fait que n'importe quel véhicule a potentiellement la possibilité de collecter des données au delà de son voisinage direct et ce, en utilisant uniquement des communications inter-véhicules. La deuxième propriété mise en évidence est sa tolérance vis-à-vis des partitionnements du réseau. Enfin, la troisième propriété repose sur le fait que la collecte de données, via le réseau, s'effectue à la demande et se termine une fois que cette dernière est achevée, menant en particulier à des économies de bande passante. À notre connaissance, c'est le premier protocole révélant simulatanément ces trois caractéristiques. Outre la preuve de validité théorique, notre protocole a été implémenté puis testé au travers de la suite logicielle Airplug, conduisant à des expérimentations sur routes ainsi qu'à une évaluation des performances.*

ABSTRACT. *In this paper, we present a protocol to collect data within a vehicular ad hoc network (VANET). In spite of the intrinsic dynamicity of such network, our protocol simultaneously offers three relevant properties: (1) It allows any vehicle to collect data beyond its direct neighborhood (i.e., vehicles within direct communication range) using vehicle-to-vehicle communications only (i.e., without passing through any infrastructure or one of its components); (2) It tolerates possible network partitions; (3) It works on demand and stops when the data collection is achieved. To the best of our knowledge, this is the first collect protocol having these three characteristics. In addition to a theoretical proof of correctness, our protocol has been implemented and tested through the Airplug Software Distribution: Road and lab experiments are presented and discussed.*

MOTS-CLÉS : *Réseau ad hoc de véhicules ; collecte de données ; communications inter-véhicules*

KEY WORDS: *VANet; data collection; inter-vehicle communication*

1re soumission à *CFIP 2011*, le 14 février 2011.

1. Introduction

1.1. *Contexte*

Accompagnant l'évolution de l'industrie automobile, les véhicules sont de plus en plus équipés de capteurs, dans le but d'améliorer à la fois la sécurité des passagers ou le confort de conduite. Prises individuellement, les informations locales provenant des différents capteurs d'un même véhicule fournissent le plus souvent une connaissance imparfaite, tronquée ou bien incomplète de l'environnement relatif à la route. Cependant, en confrontant les informations issues de plusieurs véhicules, il devient possible d'étoffer et de compléter la perception qu'ont les véhicules de leur environnement. En d'autres termes, en échangeant et en comparant les informations émanant de diverses automobiles, l'information devient plus précise et plus pertinente : elle autorise un degré de confiance beaucoup plus élevé. En particulier, il a été démontré que les incertitudes relatives à une localisation collaborative dans un groupe d'agents ou de véhicules est moindre que dans le cas où les agents s'essayent à estimer leurs positions indépendamment des acquisitions effectuées par leurs semblables [CAP 02, ROU 04].

Aujourd'hui, la littérature offre un large panel de techniques pour intégrer les observations locales : filtres de Kalman, simulations de Monte-Carlo, estimation du maximum de vraisemblance [KAR 04, PAR 07, MO 09]... Néanmoins, bien que ces stratégies aient démontrées leur efficacité au travers de maintes simulations et d'expérimentations, elles supposent toujours que les informations ont déjà été collectées et sont donc disponibles à souhait. Autrement dit, le processus consistant préalablement à collecter les données est le plus souvent occulté. En effet, via un opérateur de téléphonie mobile, on peut facilement envisager d'envoyer et de recevoir des données collectées à une large échelle : nombreuses sont alors les applications possibles telles que l'estimation du traffic routier, le nombre de places disponibles sur un parking, la vitesse moyenne sur un itinéraire donné, etc. Toutefois, bien qu'une simple connexion 3G permette d'envoyer des données issues des véhicules, il semble généralement plus pertinent d'agréger ces données avant de les envoyer.

En effet, ce *modus operandi* facilite l'analyse et le filtrage contextuel en vue d'envoyer uniquement les informations pertinentes et ce, à des fins d'économie de bande passante voire de post-traitement. Dans ce contexte, il est donc nécessaire de combiner l'envoi des données avec un processus de collecte. C'est ce dernier point, à savoir la collecte de données au sein d'un réseau de véhicules, qui fait l'objet de notre article.

1.2. *État de l'art*

La plupart des collectes de données réalisées dans les réseaux véhiculaires le sont par l'intermédiaire d'un mécanisme de dissémination par lequel chaque véhicule diffuse périodiquement ses propres informations à travers le réseau. La littérature ayant trait au réseau de véhicules présente un large éventail de protocoles de dissémination. On peut par exemple se référer aux disséminations dites *opportunistes* [WU 04] où les messages sont envoyés à tout véhicule rencontré sur le trajet. On peut également citer les disséminations géographiques [LEE 06] pour lesquelles les informations sont propagées vers les noeuds avoisinants les plus proches. On peut aussi faire mention des disséminations pair-à-pair [LEE 06] ou de celles basées sur des grappes de véhicules [BON 07].

Néanmoins, toutes ces disséminations et, par extension, toutes les collectes de données reposant sur ces dernières, ne résultent pas d'une demande spécifique correspondant à un besoin momentané. En conséquence, les informations sont diffusées régulièrement et de manière récurrente même si elles ne sont pas utilisées. Pire encore : puisque les véhicules ne savent pas quelles sont les données qui pourraient servir, ils auront souvent tendance à propager plus de données que nécessaire, menant ainsi à un gaspillage de la bande passante.

Pour contourner le problème, il serait donc plus judicieux que toute collecte de données découle d'une requête correspondant à un besoin précis émanant d'un véhicule spécifique. Dans un réseau fixe, une telle collecte peut être facilement mise en pratique grâce à un algorithme à vague dont le PIF (*Propagation of Information with Feedback*) est probablement l'exemple le plus emblématique [SEG 83]. Sans entrer dans les détails, l'algorithme du PIF fonctionne en deux étapes, chacune d'entre elles s'apparentant au sac et au ressac de la mer (d'où le nom d'*algorithme à vague*). Ces étapes consistent en une phase de diffusion d'un message suivie ensuite d'une phase d'accusé de réception, phases durant lesquelles le maintien d'une structure virtuelle correspondant à un arbre couvrant est requis. Malheureusement, le maintien de cette structure est un prérequis incontournable pour s'assurer du bon fonctionnement de l'algorithme, ce qui ne peut être garanti dans un réseau dont la topologie est sujette à des changement incessants.

Dans [CHE 02], les auteurs tentent d'outrepasser ce problème en adaptant pour les véhicules un algorithme à vagues décentralisées issue de [FIN 79]. Malgré cela, leur protocole repose sur l'hypothèse relativement forte que le réseau conserve sa connexité en dépit de sa dynamique, inhérente aux réseaux de véhicules. En particulier, il est posé comme condition *sine qua non* qu'aucun véhicule ne peut disparaître ou sortir du réseau, ce qui arrive pourtant fréquemment au sein des VANETs.

1.3. *Contribution*

Notre contribution s'articule autour des deux points suivants.

1) *Description d'un protocole de collecte auto-stabilisant.* Nous présentons un protocole de collecte pour réseaux de véhicules en mettant en jeu uniquement les communications inter-véhicules. Dans notre schéma, toute collecte de données émane préalablement d'une demande incluant, de manière non exhaustive, la durée et la profondeur de la collecte ainsi que le type de données visé par cette dernière. Contrairement à [CHE 02], il n'est pas impératif que le réseau demeure toujours connexe. Pour parvenir à cela, chaque véhicule est amené à confronter sa vue locale du réseau avec celles des autres pour la maintenir à jour. Un r-opérateur est utilisé pour cela. Il s'agit d'un outil emprunté au domaine de l'auto-stabilisation qui permet à notre protocole de supporter les changement de topologie du réseau. En particulier, toutes les vues locales tendront à se corriger d'elles-mêmes en dépit des perturbations sous-jacentes. Faute de place, nous ne détaillons pas les concepts d'auto-stabilisation et de r-opérateurs. Pour plus de détails, le lecteur pourra se reporter aux articles [DUC 01, BRU 07, DUC 10b].

2) *Implémentation et expérimentations sur route ainsi qu'en laboratoire.* Via la suite logicielle Airplug [DUC 09, KHA 10] qui est un ensemble de programmes facilitant le développement de solutions distribuées dédiées aux réseaux dynamiques, nous avons effectué plusieurs expériences sur route et en laboratoire. Ces expérimentations ont confirmé la tolérance de notre protocole face aux

partitionnements du réseau. Elles ont également mis en lumière l'influence de certains paramètres selon que le réseaux est faiblement ou fortement dynamique.

2. Protocole COL

Dans cette section, nous présentons l'algorithme de collecte COL, qui se base sur la notion de vue locale et sur le r-opérateur `ant`. Avant d'en donner les détails, nous présentons tout d'abord l'intuition sous-jacente.

2.1. *Intuition*

Dès l'instant où il se retrouve impliqué dans un processus de collecte, tout véhicule se met à diffuser périodiquement sa vue locale à son voisinage. La vue locale, de profondeur j, est simplement une liste $(N_0, N_1, \ldots, N_p)$ telle que pour tout $j \in \{0, 1, \ldots, p\}$, N_j est l'ensemble des couples (v, x_v) où v est un véhicule se situant à j sauts de l'initiateur et où x_v correspond à la donnée locale de v visée par la collecte.

Au départ de chaque collecte, seul l'initiateur est impliqué dans le processus de collecte. Mais le nombre de véhicules concernés augmente au fur et à mesure de la propagation des messages issus de l'initiateur à travers le réseau. Cependant, nous bornons l'exploration du réseau à `maxdst` (*maximal distance*) sauts de l'initiateur.

Parallèlement à cela, chaque véhicule remet périodiquement sa propre vue locale à jour en fonction des autres vues qu'il a reçues et ce, en appliquant le r-opérateur `ant`. Cette opération, répétée à chaque expiration de timer (de valeur `aTimer`), nous assure que d'éventuels nouveaux véhicules impliqués dans la collecte ainsi que les possibles changements topologiques finissent par être pris en compte par chacun des véhicules. En particulier, les vues locales devenues obsolètes au gré de la dynamique auront tendance à se corriger d'elles-mêmes par l'intermédiaire du r-opérateur `ant` et de sa propriété intrinsèque d'autostabilisation.

L'opérateur `ant` consiste d'abord à intercaller un ensemble vide au début des vues reçues et ensuite à les fusionner avec la vue locale terme à terme en supprimant l'information redondante. Par exemple, si la vue du véhicule 1 est $\mathcal{V}_1 = (\{(1,a)\}, \{(2,b),(3,c),(4,d)\})$ et s'il reçoit la vue $\mathcal{V}_2 = (\{(3,c)\}, \{(1,a),\ (4,d),(5,e)\}, \{(2,b),(6,f),(8,h),(7,g)\})$, alors il calculera la vue $ant(\mathcal{V}_1, \mathcal{V}_2) = (\{(1,a)\}, \{(2,b),(3,c),(4,d)\}, \{\ \sout{(1,a)},\ \sout{(4,d)}, (5,e)\}, \{\ \sout{(2,b)}, (6,f),(8,h),(7,g)\})$ = $(\{(1,a)\}, \{(2,b),(3,c),(4,d)\}, \{(5,e)\}, \{(6,f),(8,h),(7,g)\})$.

La terminaison de la collecte est déterminée de deux manières. D'une part, chaque véhicule quitte le processus de collecte apres `maxdur` (*maximal duration*) expirations de timer, cessant alors de mettre à jour et d'émettre sa vue locale. D'autre part, si un véhicule fabriquait `maxstb` (*maximal stability*) vues successives identiques, il quitterait également la collecte prématurément, ceci pour éviter d'attendre une consolidation de la vue dans un réseau devenu stable (à un feu rouge par exemple).

Notons qu'il convient d'assurer `maxdst` $\leq$ `maxdur` pour que l'exploration du réseau soit possible jusqu'à `maxdst` sauts de l'initiateur. De même, on devrait avoir `maxstb` $<$ `maxdur` pour obtenir une

convergence plus rapide en cas de réseau stable. Lorsque la collecte est terminée, c'est la vue locale de l'initiateur qui est considérée comme étant le résultat. Elle contient les différentes données classées selon la distance à l'initiateur, ce qui est une information riche pour produire une aggrégation des données : simple moyenne, moyenne pondérée en fonction de la distance, etc. Ce résultat est ensuite exploité localement par l'initiateur, qui peut aussi l'envoyer à d'autres véhicules ou vers un serveur de l'infrastructure via une passerelle [DUC 10a].

2.2. *Gestion du voisinage dynamique*

Afin de gérer au mieux l'instabilité du voisinage, chaque fois qu'un véhicule `v` reçoit un message d'un véhicule `u`, il lui alloue localement une durée de vie égale à `maxloss` timers (lignes 17 et 26). De cette manière, si `v` ne reçoit aucun autre message de la part de `u` pendant `maxloss` timers, `v` considèrera que `u` ne fait plus partie de son voisinage. Plus précisément, à chaque expiration du timer, la durée de vie de `u` est décrémentée de 1 (ligne 31), excepté, bien entendu, si `u` a renvoyé un autre message à `v` (auquel cas la durée de vie de `u` est remise à `maxloss`). Lorsque la durée de vie atteint la valeur de 0, toutes les données relatives à `u` sont effacées de `v` (ligne 33).

2.3. *Collecter des données*

2.3.1. *Débuter une collecte*

Pour des raisons de simplicité, nous supposerons dans ce qui suit que toute collecte de données sera déclenchée par un seul et même véhicule appelé l'initiateur. Dès l'instant que l'initiateur décide de débuter une collecte de données, il envoie un message composé de quatre champs à destination de son voisinage (ligne 12). Ces champs correspondent respectivement au numéro de la collecte (col_id), l'identifiant de l'initiateur (col_initiator), les paramètres de la collecte (col_param) et sa vue locale (col_view).

Détaillons ces champs. Concernant l'identifiant de l'initiateur on suppose pour des raisons de simplicité que celui-ci demeure inchangé, néanmoins notre protocole pourrait sans soucis autoriser l'exécution simultanée de plusieurs collectes initiées par différents véhicules. Inversement, le numéro de la collecte est quant à lui incrémenté de 1 à chaque nouvelle collecte de données. En ce qui a trait aux paramètres de la collecte, ces derniers sont au nombre de 4 : `typedt` (le type de données à collecter), `maxdst` (la profondeur maximale de la collecte exprimée en nombre de saut depuis l'initiateur), `maxdur` (la durée maximale de la collecte exprimée en nombre d'expiration de timers) et `maxstb` (le nombre maximal de vues successives stables avant terminaison locale).

On remarquera qu'au moment de la première émission d'un message par l'initiateur, sa connaissance sur son voisinage est réduit à l'ensemble vide. En particulier, sa vue locale ne contient que son identifiant accompagné de sa donnée locale visée par la collecte. Toutefois, sa vue locale et sa connaissance du voisinage vont aller en s'étoffant au fur et à mesure de l'avancement de la collecte.

Algorithme de collecte COL, exécuté par un véhicule v

1 Starting_action(`typedt`, `maxdst`, `maxdur`, `maxstb`) :
▷ *Par simplicité, nous supposons qu'il existe un seul initiateur. Les autres noeuds ne peuvent pas lancer de collecte.*
2 **si** v n'est pas l'initiateur ou col_active == vrai **alors sortir**
3 col_active ← vrai ▷ *Une collecte est en cours*
4 col_id += 1 ▷ *Identifiant de la collecte est incrémenté*
5 col_initiator ← v
6 col_param ← (`typedt`, `maxdst`, `maxdur`, `maxstb`)
7 local_data ← la donnée locale de v de type `typedt`
8 local_view ← $\{(v, \text{local\_data})\}$
9 count_dur ←0 ▷ *Compte jusqu'à `maxdur`*
10 count_stb ← 0 ▷ *Compte jusqu'à `maxstb`*
11 tab_views ← $\emptyset$ ▷ *Liste des dernières vues locales reçues*
12 **envoyer**(col_id, col_initiator, col_param, local_view)
13 armer le timer avec une durée égale à `aTimer`

14 À la réception d'un message :
15 **recevoir**(rcv_id, rcv_init, rcv_par, rcv_view) de u
16 **si** col_active == vrai **et** col_id == rcv_id **alors**
▷ *Message pour la collecte en cours*
17 tab_lifetime[u] ← `maxloss`
18 tab_views[u] ← rcv_view ▷ *Sauvegarder la vue reçue*
19 **sinon si** rcv_id > col_id
▷ *New collect*
20 col_active ← vrai ▷ *Une collecte est localement en cours*
21 col_number ← rcv_id ▷ *Sauvegarde de l'identifiant de la collecte*
22 col_initiator ← rcv_init
23 (`typedt`, `maxdst`, `maxdur`, `maxstb`) ← rcv_par
24 count_dur ←0 ▷ *Compte jusqu'à `maxdur`*
25 count_stb ← 0 ▷ *Compte jusqu'à `maxstb`*
26 tab_lifetime[u] ← `maxloss`
27 tab_views[u] ← rcv_view ▷ *Sauvegarde de la vue locale reçue*
28 armer le timer avec une durée égale à `aTimer`
29 **fin si**
▷ *Les autres messages sont ignorés.*

30 À l'expiration d'un timer :
▷ *Détection de la disparition de voisins*
31 tab_lifetime[u] -= 1 Pour tout u dans tab_lifetime
32 **pour chaque** u tel que lifetime[u] == 0 **faire**
33 Effacer toutes les entrées de u dans tab_lifetime et tab_views
34 **fin pour**
▷ *Calculer la nouvelle vue locale*
35 old_local_view ← local_view

```
36  local_data ← la donnée locale de v de type typedt
37  local_view ← {(v, local_data)}
38  pour chaque u tel que tab_views[u] existe faire
39      local_view ← ant(local_view, tab_views[u])
40  fin pour
41  Tronquer local_view en ne gardant que les maxdst premiers éléments
    ▷ Détection de la terminaison
42  count_dur += 1
43  si old_local_view et local_view sont équivalentes alors
44      count_stb += 1
45  sinon
46      count_stb ← 0
47  fin si
48  col_active ← faux
49  si col_initiator ∈ local_view alors
        ▷ Vue valide au regard de la collecte
50      envoyer( col_id, col_initiator, col_param, local_view )
51      si count_stb ≠ maxstb et
                              count_dur ≠ maxdur alors
52          réarmer le timer avec une durée égale à aTimer
53          col_active ← vrai
54      sinon si col_initiator == v
            ▷ Fin de la collecte sur l'initiateur.
55          Calcul du résultat final en utilisant la vue locale local_view
56          col_active ← faux                    ▷ Permet de débuter une nouvelle collecte
57      fin si
58  fin si
```

2.3.2. *À la réception d'un message*

À l'arrivée d'un message (ligne 15), le véhicule v s'assure que ce dernier est pertinent en regardant le numéro de la collecte rcv_id qui lui est associé. Pour parvenir à cela, rcv_id est comparé à col_id qui correspond au numéro de la dernière collecte auquelle v a pris part. Nous avons trois cas à considérer.

Dans le premier cas, si v est impliqué dans une collecte (col_active est à vrai) et si rcv_id et col_id sont de même valeur, alors le message est considéré comme devant être pris en compte pour la collecte qui est en cours (ligne 16).

Pour le second cas, si v est impliqué dans une collecte et que le message reçu est associé à une nouvelle collecte (i.e., rcv_id > col_id, ligne 19), le véhicule réinitialise ses variables de contrôle associées à la collecte à laquelle il participait. Cela peut par exemple survenir lorsque l'initiateur a mis fin prématurément à une collecte (e.g., lorsque la vue locale est restée stable pendant suffisamment longtemps) et en a redémarrée une autre dans la foulée.

En ce qui concerne le troisième cas où `v` n'est pas impliqué dans une collecte (col_active est à `faux` sur `v`), le message est pris en compte seulement s'il résulte d'une nouvelle collecte, à savoir rcv_id $>$ col_id (ligne 19). De cette façon, on ignore les messages provenant d'anciennes collectes, devenus obsolètes.

Lorsqu'un message est reconnu comme étant pertinent, la vue locale située dans le dernier champs du message est stockée localement (lignes 18 et 27). De plus, à la réception du premier message associé à une collecte spécifique, les paramètres inherents à cette dernière sont conservés localement (jusqu'au terme de la collecte) et les variables de contrôle ainsi que le timer sont réinitialisés (ligne 28).

2.3.3. *À l'expiration du timer*

À chaque expiration du timer, un véhicule `v` met à jour sa vue locale du réseau. Celle-ci est calculée en fonction uniquement des vues locales des voisins considérés comme étant encore présents dans le voisinage (i.e., les véhicules pour lesquels la durée de vie n'est pas tombée à zéro).

Pour parvenir à cela, il est fait usage du r-opérateur ant (ligne 39) afin d'obtenir une nouvelle vue locale, qui sera tronquée après les `maxdst` premiers éléments et ce, en vue de respecter le critère de la distance maximale de la collecte (ligne 41).

Une fois l'actualisation de la vue locale achevée, se pose alors la question de la terminaison de la collecte (lignes 42-58). Le nombre d'expirations de timers est comptabilisé et incrémenté depuis le début de la collecte en cours (ligne 42) par le biais de la variable count_dur. La variable count_stb, quant à elle, correspond au nombre de timers successifs ayant produits la même vue (lignes 43-47). Par ailleurs, si jamais l'initiateur n'apparaît pas dans la vue locale de `v`, alors on considère que ce dernier n'est pas ou plus concerné par la collecte car cela signifie en particulier qu'il se situe à une distance supérieure à `maxdst` de l'initiateur. Dans le cas contraire, la nouvelle vue est diffusée dans le voisinage, avec les paramètres de la collecte.

Enfin, si l'une des deux variables count_dur et count_stb n'a pas atteint sa valeur maximale, respectivement `maxdur` et `maxstb` (ligne 51), le timer est à nouveau réarmé en vue d'une nouvelle mise à jour de la vue lors de sa prochaine expiration (ligne 52). Dans le cas contraire, la collecte est localement terminée. Le résultat de la collecte correspond à la vue locale de l'initiateur lorsqu'il a localement terminé.

3. Expérimentations sur route et en laboratoire

3.1. *Suite logicielle Airplug*

Nous avons réalisé nos expérimentations avec la suite logicielle Airplug, que nous décrivons brièvement dans cette section.

Airplug est un intergiciel de communication léger pour réseaux dynamiques [DUC 09]. Il se caractérise par sa robustesse et sa simplicité dans l'organisation des échanges de messages intra- et inter-véhicules. L'architecture de Airplug repose sur les facilités procurées par les systèmes d'exploitation standard : allocation de ressources, ordonnancement des processus, gestion du « temps-réel »,

etc. Ceci évite toute redondance entre l'intergiciel et le système d'exploitation (qui doit être POSIX). En outre, Airplug étant développé en « espace utilisateur », aucune modification n'est nécessaire au niveau du noyau, ce qui accroît sa portabilité.

Le programme-coeur Airplug s'intercale entre les interfaces réseaux et les applications, ce qui permet de se prémunir contre une application qui consommerait trop de ressource (e.g. application tiers boguées). Toutes les communications via Airplug se font par passage de messages. Un message provenant d'une application peut être envoyé à plusieurs applications, locales ou distantes. Cependant, une application ne peut pas recevoir les messages adressés à toutes les applications, ni émis par une application non locale sans s'être abonnée auprès d'Airplug à l'application émettrice. Ce principe d'abonnement (confiance relative en local, confiance limitée à distance) permet à une application de contrôler ses réceptions. En outre, cela augmente la robustesse de l'architecture en évitant les problèmes en cascade dans le cas d'applications boguées (sur lesquelles il est plus facile d'agir lorsqu'elles sont locales que lorsqu'elles sont distantes).

Les messages utilisent un format d'adressage spécifique. La destination d'un message est composée de deux champs : un champ zone qui contient la zone d'envoi du message suivi du nom de l'application destinataire. Cet adressage simplifié permet de supporter la dynamique du réseau et s'avère néanmoins suffisante pour construire protocoles de communication et applications réparties. La zone peut être interne (LCH pour localhost) ou externe (AIR), c'est-à-dire composée des voitures dans le voisinage, ou bien les deux (ALL). Mais elle peut également être plus spécifique (nom ou adresse d'une voiture avoisinante).

Chaque application ou protocole développé pour Airplug s'exécute dans un processus dédié, qui est lancé par Airplug. Le développement de nouveaux protocoles de communication est donc simplement réalisé en espace utilisateur. Les communications inter-processus sont réalisées en utilisant les entrées et sorties standard, qui sont connectées à Airplug à l'aide de *pipe*. Cette méthode simple et robuste permet de développer des protocoles en utilisant tout type de langage de programmation. Comme les interfaces réseaux sont également gérées par Airplug, les applications accèdent au réseau de la même manière qu'elles le feraient pour communiquer avec d'autres applications locales, simplement en écrivant sur leur sortie standard. Airplug reçoit les données envoyées par les processus et les envoie vers l'interface désirée.

Cette technique de communication inter-processus a permis de développer un émulateur de réseaux de véhicules basé sur les facilités du shell [KHA 10]. Grâce à cet émulateur, les applications utilisées sur route s'exécutent sur un seul ordinateur. Les connexions sont réalisées à l'aide de tubes nommés, qui sont créés ou détruits selon des fichiers de position. Ces fichiers émanent des expérimentations sur route (positions GPS), de Network Simulator ou d'autres générateurs de trafic. L'émulateur permet de jouer avec les paramètres de dynamique du réseau, de fiabilité des liens, de portée des communications sans fil, etc.

3.2. *Expérimentation sur route*

Pour valider notre protocole, nous l'avons implémenté puis testé sur route en utilisant la suite logicielle Airplug. Pour cela, 5 véhicules ont été mobilisé. Chacun d'eux comprenait exactement un PC (Dell mini-9 Modèle DP118) sous Ubuntu (v8.04 Hardy Heron) exécutant Airplug ainsi que

notre protocole COL. Tous les PCs sont équipés d'une carte WiFi externe à connectique USB (Alfa AWUS036EH) permettant de connecter une antenne sur le toit des véhicules (D-link ANT24-0700).

Lors de cette expérimentation, nous avons considéré un scénario correspondant à un convoi de 5 véhicules roulant à une vitesse d'environ 90 km/h. Le véhicule en tête du convoi jouait le rôle de l'initiateur en lançant successivement des collectes répondant aux paramètres suivants : `aTimer` = 1000 ms, `maxloss` = 2, `typedt` = identité, `maxdst` = 4 et `maxdur` = 10.

La durée d'un timer ainsi que la valeur de `maxloss` ont été fixé avec une valeur suffisamment grande pour que des messages puissent être échangés et que les vues locales ne soient pas mises à jour incessamment. Le type de données à collecter était l'*identité* de manière à vérifier le nombre de véhicules atteints par la collecte. En ce qui concerne `maxdst`, il a été fixé à 4 de façon à ce que tous les véhicules du convoi puissent être impliqués. Le paramètre `maxdur` a été fixé comme étant égal à 10 expirations de timers. De cette manière, la durée de la collecte est suffisamment longue pour éventuellement permettre à des données issues du dernier véhicule de remonter en tête du convoi pour atteindre l'initiateur.

Pour cette expérience, les résultats ont été plus que concluants puisque l'initiateur a toujours été à même de récupérer toutes les données du convoi, malgré les modifications de la topologie. Cette preuve de concept démontre l'intérêt de notre protocole en pratique.

3.3. *Expérimentations en laboratoire*

Nous avons réalisé des séries de 50 expériences en laboratoire grâce à l'émulateur de réseaux, en reprenant le scénario d'un convoi constitué cette fois-ci de 13 véhicules et en faisant varier la fiabilité des liens (cf. figure 1.a) et la durée des timers `aTimer` (cf. figure 1.b).

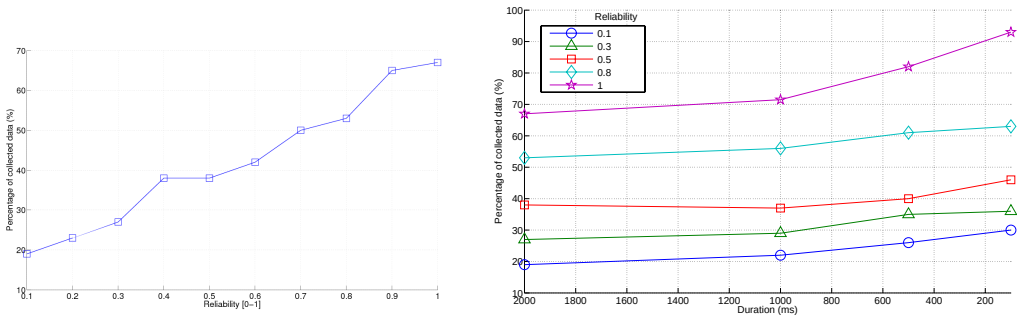

Figure 1. *Expérimentations avec l'émulateur de réseaux de véhicules : pourcentage de données collectées en fonction de la fiabilité des liens (à gauche) et de la durée du timer (à droite).*

Comme l'on pouvait s'y attendre, plus les liens sont fiables, plus le nombre de données collectées est grand. On constate également que la durée du timer a une faible influence sur les résultats sauf dans le cas de liens fiables. En effet, dans ce dernier cas, c'est essentiellement la dynamique du réseau qui gêne la collecte. Or, un timer court permet de collecter rapidement les données, avant que la

topologie ne change. Notons qu'un timer court peut contribuer à une forte occupation (momentanée) du réseau. Une valeur trop faible n'est pas forcément un avantage. Une valeur de 1 s semble être un bon compromis

D'autres expérimentations ont été menées en faisant varier le paramètre `maxloss`. Une valeur élevée permet de sauvegarder plus longtemps les données ayant été à un moment ou à un autre dans le voisinage. Cependant, plus la valeur `maxloss` est grande moins les données collectées sont mises à jour, ce qui peut être par exemple un problème dans le cadre d'une collecte de vitesses ou de positions. À titre d'exemple, pour notre scénario en convoi, une valeur `maxloss` = 2 était suffisante.

4. Conclusion

Dans cet article, nous avons présenté un protocole de collecte de données pour réseaux fortement dynamiques, telles que les VANETs. Ce protocole peut fonctionner en utilisant uniquement des communications inter-véhicules, c'est-à-dire sans passer par l'intermédiaire d'une infrastructure fixe. Notre protocole est basé sur l'opérateur ant, qui permet de construire une vue locale du réseau et ainsi de collecter les données malgré les changements de topologie. En particulier, et contrairement aux protocoles existants dans la littérature, notre algorithme continue de fonctionner même si le réseau subit des partitionnements.

Des expérimentations ont été menées avec l'émulateur de réseaux de véhicules de la suite Airplug afin d'évaluer son comportement sous diverses valeurs des paramètres. L'algorithme COL a également été testé sur route. Il a été utilisé avec succès pour collecter la moyenne des vitesses d'un convoi de véhicules, en le combinant avec une découverte de connexion Internet [DUC 10a]. Un film ainsi qu'une copie d'écran de l'émulateur sont disponibles en ligne à l'adresse http ://www.hds.utc.fr/airplug.

C'est en évitant de maintenir une structure virtuelle dans le réseau que notre algorithme peut fonctionner malgré la dynamique du réseau. Nous envisageons d'autres travaux dans cette voie.

5. Bibliographie

[BON 07] BONONI L., FELICE M. D., « A Cross Layered MAC and Clustering Scheme for Efficient Broadcast in VANETs », *IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems*, 2007, p. 1-8.

[BRU 07] BRUKMAN O., DOLEV S., HAVIV Y. A., YAGEL R., « Self-Stabilization as a Foundation for Autonomic Computing », *ARES*, 2007, p. 991-998.

[CAP 02] CAPKUN S., HAMDI M., HUBAUX J.-P., « GPS-free Positioning in Mobile Ad Hoc Networks », *Cluster Computing*, vol. 5, n° 2, 2002, p. 157-167.

[CHE 02] CHEN S.-H., HUANG T.-L., « A Wave Algorithm for Mobile Ad Hoc Networks », *Workshop on Algorithms and Computational Molecular Biology co-located with ICS*, 2002.

[DUC 01] DUCOURTHIAL B., TIXEUIL S., « Self-stabilization with r-operators », *Distributed Computing*, vol. 14, n° 3, 2001, p. 147-162.

[DUC 09] DUCOURTHIAL B., KHALFALLAH S., « A Platform for Road Experiments », *VTC Spring*, 2009.

[DUC 10a] DUCOURTHIAL B., ELALI F., « A light architecture for opportunistic vehicle-to-infrastructure communications », *ACM International Symposium on Mobility Management and Wireless Access*, 2010.

[DUC 10b] DUCOURTHIAL B., KHALFALLAH S., PETIT F., « Best-effort group service in dynamic networks », *SPAA*, 2010, p. 233-242.

[FIN 79] FINN S., « Resynch Procedures and a Fail-Safe Network Protocol », *IEEE Transactions on Communications*, vol. 27, n° 6, 1979, p. 840-845.

[KAR 04] KARAMI E., SHIVA M., « Maximum likelihood MIMO channel tracking », *VTC*, 2004, p. 876-879.

[KHA 10] KHALFALLAH S., DUCOURTHIAL B., « Bridging the Gap between Simulation and Experimentation in Vehicular Networks », *VTC Fall*, 2010, p. 1-5.

[LEE 06] LEE U., MAGISTRETTI E., ZHOU B., GERLA M., BELLAVISTA P., CORRADI A., « Efficient Data Harvesting in Mobile Sensor Platforms », *PerCom Workshops*, 2006, p. 352-356.

[MO 09] MO Z., ZHU H., MAKKI K., PISSINOU N., KARIMI M., « On Peer-to-Peer Location Management in Vehicular Ad Hoc Networks », *International Journal of Interdisciplinary Telecommunications and Networking (IJITN)*, vol. 1, n° 2, 2009, p. 28-45.

[PAR 07] PARKER R., VALAEE S., « Vehicular Node Localization Using Received-Signal-Strength Indicator », *IEEE Transactions on Vehicular Technology*, vol. 56, 2007, p. 3371-3380.

[ROU 04] ROUMELIOTIS S., REKLEITIS I., « Propagation of uncertainty in cooperative multirobot localization : Analysis and experimental results », *Autonomous Robots*, vol. 17, n° 1, 2004, p. 41-54.

[SEG 83] SEGALL A., « Distributed network protocols », *IEEE Transactions on Information Theory*, vol. 29, n° 1, 1983, p. 23-34.

[WU 04] WU H., FUJIMOTO R. M., GUENSLER R., HUNTER M., « MDDV : a mobility-centric data dissemination algorithm for vehicular networks », *Vehicular Ad Hoc Networks*, 2004, p. 47-56.