Algorithme "trickle" [1] pour les réseaux de capteurs sans fil

Mathilde Arnault (CEA)- Paolo Ballarini (CS) - Boutheina Bannour (CEA) - Pascale Le Gall (CS)

Date de soutenance : 27 mars 2018

1 Objectif général

Le présent document décrit un projet associé au cours MVS (modélisation et vérification des systèmes temps réels). Il s'agit de l'étude d'un algorithme, appelé Trickle, qui a pour objectif de partager de l'information au sein d'un réseau.

L'algorithme Trickle est largement utilisé dans les réseaux de capteurs et il continue à être étudié, afin d'en connaître ses performances ou bien d'en dériver une variante ou encore de l'utiliser dans des applications de plus haut niveau (par exemple un protocole).

Le retour attendu du projet est double

- la modélisation et l'analyse de l'algorithme Trickle à l'aide des outils UPPAAL et Diversity, ainsi qu'à l'aide de simulations directes (développées de façon adhoc). A travers une utilisation combinée ou séparée des approches (UPPAAL, Diversity, simulation), l'objectif est d'acquérir une expertise fine de l'algorithme Trickle (i.e. atouts et inconvénients selon le contexte d'utilisation) ;
- au-delà de l'étude de l'algorithme Trickle, l'objectif est d'acquérir des compétences pour utiliser à bon escient les outils d'analyse (type UPPAAL et Diversity) selon les besoins.

De plus, dans une approche du travail mise en avant par le *Génie Logiciel*, la réalisation du projet sera menée selon une démarche coopérative :

- un sous-groupe (A par la suite) en charge de la modélisation UPPAAL ;
- un sous-groupe (B par la suite) en charge de la modélisation Diversity ;
- un sous-groupe (C par la suite) en charge de la réalisation d'une simulation (implémentée dans le langage de programmation de leur choix).

Les sous-groupes A, B, C seront appelés à coopérer :

- pour s'assurer que les hypothèses de modélisation sont identiques, ou au contraire, pour poser des hypothèses différentes en lien avec les atouts des approches (UPPAAL, Diversity, simulation) ;
- pour confronter les approches en analysant ou exécutant des données issues d'une approche avec une autre approche.

Une réalisation du projet bâtie sur la coopération des sous-groupes devra être privilégiée.

L'évalution du projet se fera par le biais d'une soutenance organisée en 7 parties : A, B, C, AB, BC, AC, ABC (A pour une présentation du sous-groupe A, AB pour une présentation commune du travail coopératif entre les sous-groupes A et B, ABC pour une présentation commune à l'ensemble des sous-groupes A, B et C).

La partie écrite du rendu du projet sera fait par sous-groupe et se présentera sous la forme d'un répertoire compressé comprenant, à tître indicatif, l'ensemble des fichiers suivants :

Un fichier readme.txt qui liste l'ensemble des fichiers du répertoire en indiquant le rôle de chacun des fichiers ou répertoires présents ;

1. Traduction proposée : *Algorithme de ruissellement ou Algorithme "filet d'eau"*.

Le ou les fichiers de code source (par défaut, chaque fichier source contient un commentaire entête indiquant le rôle du fichier, l'auteur et les dates de création, et de révisions successives, et il est attendu que le code soit commenté) ;

Le ou les fichiers de données utilisées ainsi que le ou les fichiers de résultats, pour peu qu'ils présentent un intérêt ;

Un mode d'emploi simple pour exécuter les programmes, sur vos données de test (faire en sorte qu'une commande sans arguments permette d'exécuter vos programmes et de fournir les résultats assortis de quelques commentaires) ;

Un document rédigé, en général sous format pdf comprenant les informations suivantes : l'architecture générale du projet (à savoir l'organisation des principales fonctions, types de données, fichiers) ; les principaux choix de conception effectués ; les limitations adoptées dans le projet ; les réponses aux questions directement incluses dans l'énoncé ; des instructions claires d'utilisation du code ; des éléments de validation (des cas d'utilisation - ou tests - illustrant de bon fonctionnement du projet) ; des illustrations, etc.

sans oublier d'y inclure des éléments concernant le travail inter-groupes.

L'exploitation de ressources externes (livres, article, internet, etc) est autorisée sous réserve d'indiquer les sources.

2 Description générale

Les réseaux de capteurs sans fil sont composés d'un grand nombre de capteurs avec des ressources limitées (énergie, mémoire, calcul) et appelés à fonctionner en autonomie pendant de grandes périodes (de l'ordre de l'année). Pour satisfaire aux nouveaux besoins liés à l'évolution des fonctionnalités au cours du temps, il est nécessaire de propager des mises à jour logicielles (codes) à installer sur les nœuds (capteurs) du réseau.

Ces réseaux de capteurs sont de taille et topologie variables (centaines ou milliers de capteurs, qui peuvent être ajoutés ou retirés, et éventuellement mobiles). La diffusion des informations au travers du réseau se fait par des communications (à faible portée) de capteur à capteur. De plus, les communications peuvent être asymétriques : un nœud A peut envoyer des messages à un nœud B sans que l'inverse soit possible. De plus, ces capacités de communication peuvent varier au cours du temps (mobilité, apparition et disparition de capteurs, qualité de service de la communication).

Un protocole de mise à jour efficace de l'ensemble des applications logicielles des capteurs d'un réseau doit répondre à plusieurs objectifs :

— économiser le nombre et la taille de paquets (messages) échangés entre les capteurs afin d'économiser les batteries des capteurs. En particulier, lorsque les capteurs ont tous la même version du logiciel (réseau stable), alors la fréquence et la taille des messages échangés doivent être minimales [2]. A cause de la topologie variable du réseau, la mise à jour des logiciels est une activité continue, sans cesse itérée : les nœuds doivent communiquer périodiquement pour découvrir s'il y a une nouvelle version de logiciel ;

— propager rapidement les nouvelles versions de logiciels afin de minimiser les périodes durant lesquelles les capteurs ne partagent pas les mêmes versions des applications logicielles. Lorsque les capteurs découvrent que des nœuds voisins nécessitent des mises à jour, la dernière version du code doit être diffusée rapidement.

Ainsi, la principale motivation est autant que possible de a) diffuser largement l'information lorsque de nouvelles informations sont présentes, et de b) réduire les communications en l'absence de nouvelles informations.

L'algorithme Trickle est utilisé pour la propagation et la maintenance des mises à jour logicielles dans les réseaux de capteurs sans fil. L'idée générale est la suivante :

2. Ces échanges de messages doivent juste servir à s'assurer que tous les capteurs partagent la même version logicielle.

— par défaut, les nœuds Trickle (i.e. les capteurs) informent régulièrement leurs voisins quelle est la version de code installée, version identifiée par un numéro pour simplifier la présentation [3] ;
— les nœuds restent relativement silencieux, s'ils ont récemment reçu l'information que leurs voisins possèdent le même numéro de version que le leur. Chaque nœud se contente alors de diffuser de façon périodique mais espacée, le numéro de version décrivant le code dont il dispose ;
— quand un nœud apprend qu'un nœud voisin diffuse un numéro de version strictement inférieur que le sien, il diffuse alors à l'ensemble de ses voisins une mise à jour du code. Dans le cas contraire (si un voisin diffuse un numéro de version strictement supérieur au numéro du nœud), il diffuse alors son numéro de version (afin de faire savoir qu'il a besoin de recevoir une mise à jour).

Ainsi, au lieu de diffuser systématiquement et régulièrement la dernière version logicielle, l'algorithme Trickle assure qu'un nœud ne propagera une version logicielle que s'il a eu connaissance au préalable que l'un de ses voisins est doté d'une version antérieure du logiciel.

Pour assurer à la fois une propagation rapide des nouvelles versions et un minimum de maintenance, les nœuds ajustent la durée de leurs périodes d'écoute, communiquant plus souvent lorsqu'il y a une mise à jour en cours, et espaçant les interactions lorsqu'apparemment le réseau est stable.

Les nœuds Trickle sont paramétrés par
— la taille minimale I_{min} (constante) de l'intervalle d'écoute
— la taille maximum I_{max} (constante) de l'intervalle d'écoute (de la forme $I_{min} \times 2^{max}$ avec $max \geq 0$)
— la constance de redondance k avec $k \geq 1$
— la taille actuelle I (variable) de l'intervalle, initialisée à I_{min}
— le moment de la communication τ
— le compteur c (variable), initialisé à 0
— le numéro de version n courante
— le code associé associé au numéro de version

Le fonctionnement d'un nœud suit les règles suivantes :
— au temps $t = 0$, mettre c à 0 et choisir $\tau \in [I/2, I]$
— à chaque message reçu et à jour (indiquant que le voisin partage la même version du code), incrémenter c
— à chaque message reçu avec une version inférieure à la version courante du nœud, le nœud envoie son numéro de code, et son code (à l'ensemble de ses voisins)
— à chaque message reçu avec un numéro de version supérieure, il envoie son numéro de version (à l'ensemble de ses voisins)
— à chaque message reçu avec un code (de version supérieure), il met à jour son code et son numéro de version
— au temps $t = \tau$, si $c < k$, on transmet son numéro de version
— A l'expiration de l'intervalle I, I est doublé (s'il n'est pas déjà à I_{max}) si le nœud n'a pas reçu d'informations inconsistantes au cours de la période I (numéro de version différent du numéro de version courant). Sinon, I est mis à I_{min}.

La figure 1 décrit un exemple de (petit) réseau de 5 capteurs, nommés A, B, C,D et E. Le nœud A peut recevoir de l'environnement (une passerelle) une nouvelle version logicielle (avec son numéro). Le réseau est asymétrique : A peut envoyer des messages à B et D, tandis qu'il peut recevoir des messages du capteur E.

Pour compléter cette description, il est possible de consulter les articles [1] (article de référence pour l'algorithme Trickle) et [2] (modélisation de l'algorithme Trickle avec UPPAAL).

3 Travail à faire

L'enjeu est d'étudier sous quelques conditions (topologie du réseau, valeurs des constantes k, I_{min}, I_{max}) se fait la diffusion des (nouvelles) versions de code (en particulier, en termes du compromis "vitesse de propagation d'un nouveau code" et "nombre de messages échangés").

3. Le numéro abstrait des métadonnées décrivant les données caractéristiques du logiciel.

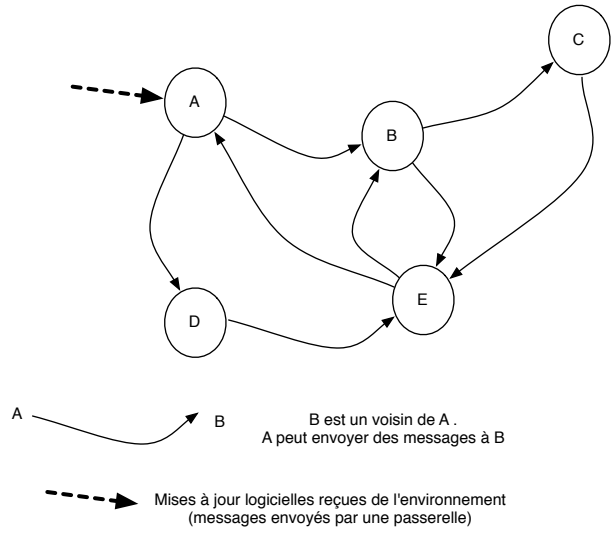

FIGURE 1 – Exemple de topologie

Il est fortement recommandé d'envisager dans un premier temps des conditions simples : un réseau réduit à quelques nœuds (3 à 5), à des valeurs simples (par exemple $I = I_{min} = I_{max}$ et $k = 1$ ou 2, $\tau = I/2$). Deux raisons à cela : une modélisation simple est plus facile à appréhender et débugguer, et les outils d'analyse type UPPAAL et Diversity font rapidement face à des problèmes d'explosion combinatoire [4].

Par ailleurs, il est fortement conseillé de (commencer par) considérer une topologie fixe.

De plus, comme tout discours en langage naturel, la description ci-dessus est susceptible d'être incomplète, sous-spécifiée, ou même parfois inconsistante à la marge. La modélisation ou implémentation amènera à expliciter des choix de conception laissés sous silence dans la description ci-dessus. Cette activité d'interprétation et de précision est très classique dans une démarche de conception pour l'ingénierie logicielle.

En vous inspirant d'un système simple PingPong à 3 nœuds dans lequel de façon cyclique, un premier nœud envoie un Ping aux deux autres nœuds qui renvoient alors un Pong en moins de 5 unités de temps ; ensuite, à la réception de 2 Pong, le premier nœud repart sur un nouveau cycle (modélisé en UPPAAL et en Diversity), il est demandé de réaliser (un item par sous-groupe) :

- une simulation de l'algorithme Trickle paramétrée par la topologie, les conditions initiales des différents nœuds (numéro de version, code associé) et les différentes valeurs des constantes ;
- une modélisation UPPAAL pour une configuration donnée (en termes de topologie, ...) ;
- une modélisation Diversity a priori pour la même configuration que pour la modélisation UPPAAL.

Lors de l'étape de conception, un **soin tout particulier** doit être accordé aux formats d'échanges de données (traces ou éléments de traces) entre les 3 modélisations, de sorte à pouvoir valider par confrontation (au moins en partie) les différentes modélisations, et à expliciter les différentes (éventuellement similaires) hypothèses sous-jacentes à la mise en œuvre.

Pour conclure, le sujet est délibéremment ouvert, et en cela, il ressemble à des activités extrêmement courantes en ingénierie logicielle. Les résultats attendus peuvent être de nature variable, selon l'approche et les objectifs qui seront privilégiés. Néanmoins, il est globalement attendu une analyse critique sur l'algorithme Trickle vis-à-vis de l'objectif d'obtention d'un état stable consistent (i.e. où tous les nœuds partagent la même version de code).

4. L'analyse de l'algorithme Trickle via des outils d'analyse est fondée sur le postulat que des propriétés établies pour des petites topologies ont toutes les chances d'être vraies pour des topologies plus grandes, un peu comme s'il y avait un principe de récurrence

4 Bibliographie

[1] Philip Levis, Neil Patel, David Culler†, Scott Shenker. Trickle : A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks.

Disponible à la page : http ://csl.stanford.edu/ pal/pubs/trickle-nsdi04.pdf

[2] Jin Song Dong, Jing Sun, Jun Sun, Kenji Taguchi, Xian Zhang. Specifying and Verifying Sensor Networks : an Experiment of Formal Methods.

Disponible à la page :
https ://pdfs.semanticscholar.org/80b6/db3367a4e6a8dee419f264dca32f7c97c7bd.pdf