WiNo : une plateforme d'émulation et de prototypage rapide pour l'ingénierie des protocoles en réseaux de capteurs sans fil

Adrien van den Bossche
CNRS-IRIT-IRT, Université de Toulouse, UT2
1 place Georges Brassens, BP 60073
31703 Blagnac Cedex, France
bossche@irit.fr

Thierry Val
CNRS-IRIT-IRT, Université de Toulouse, UT2
1 place Georges Brassens, BP 60073
31703 Blagnac Cedex, France
val@irit.fr

ABSTRACT
L'ingénierie des protocoles implique souvent de nombreuses phases de tests pour valider un protocole nouvellement créé. Dans le domaine spécifique des réseaux de capteurs sans fil, la démocratisation récente du matériel a permis de nouveaux usages pour les concepteurs de protocoles. En effet, après les phases classiques de spécification et de simulation, il est désormais courant d'évaluer le protocole dans un contexte réel et concret, sur cible finale. Dans cette optique, des outils adaptés émergent, non seulement pour rendre cette tâche plus accessible, mais aussi pour en maîtriser le déploiement, afin de rendre les résultats plus exploitables. Dans cet article, après un bref tour d'horizon des solutions disponibles, la plateforme « WiNo » d'émulation et de prototypage rapide pour réseaux de capteurs est présentée, ainsi que plusieurs exemples représentatifs de résultats obtenus grâce à son usage.

Mots-clés
Réseaux et protocoles ; Réseaux de capteurs sans fil ; WSN ; IEEE 802.15.4 ; émulation ; prototypage ; ingénierie des protocoles ; MAC ; routage

1. INTRODUCTION
Les réseaux de capteurs sans fil sont depuis quelques années au centre des thématiques de recherche de nombreux laboratoires et entreprises. Un WSN (*Wireless Sensor Network*) présente des caractéristiques particulières, par rapport à un réseau sans fil ou un réseau filaire. Le nœud est habituellement de petite taille, et dispose de faibles capacités mémoire, processeur, débit de transmission, portée radio, etc., ce qui lui permet potentiellement de répondre favorablement à une recherche de forte autonomie énergétique. Dans le même objectif, l'empilement protocolaire est lui aussi souvent réduit et se limite généralement à quatre couches, typiquement physique (PHY), liaison (MAC), réseau (NWK) et une couche application (APL) légère. La conception de protocoles pour les réseaux de capteurs n'en est pas pour autant simplifiée, et nécessite elle aussi des méthodes et des outils d'ingénierie adéquats. Nous proposons, dans ce papier, une plateforme d'émulation et de prototypage rapide « WiNo » (*Wireless Node*), adaptée à cette catégorie de réseaux, qui se veut complémentaire des outils généralement utilisés par la communauté scientifique.

Après cette introduction, le chapitre II présente un rapide état de l'art en la matière. Le chapitre III détaille l'architecture et les principes d'utilisation de la plateforme WiNo. Enfin, nous évoquons, dans le chapitre IV, les résultats obtenus par son utilisation dans le cadre de propositions de trois protocoles originaux.

2. L'EXISTANT

2.1 L'évaluation des protocoles pendant le processus de conception
L'ingénierie des protocoles nécessite une phase incontournable d'évaluation des protocoles nouvellement créés. Plusieurs outils s'offrent alors au concepteur, chacun offrant ayant ses atouts, mais aussi ses inconvénients. On cite généralement les méthodes analytiques, la simulation, l'émulation et le prototypage. Les sections suivantes rappellent les grandes lignes de chacune d'entre-elles.

2.1.1 Approches analytiques et méthodes formelles
Les approches analytiques et les méthodes formelles, comme par exemple le calcul réseau (*Network Calculus*) ou les réseaux de Petri, présentent un haut niveau d'abstraction et permettent, en se focalisant sur un point précis, d'en évaluer très précisément les capacités, les performances, etc. Cependant, il est nécessaire de replacer le dispositif étudié dans son contexte afin de pouvoir généraliser les résultats obtenus au processus entier, au protocole, voire même à tout le réseau.

2.1.2 Simulation
Par rapport aux méthodes théoriques précédentes, la simulation permet de mieux se rapprocher de la réalité en tenant compte du réseau dans une meilleure intégralité, et de prendre en compte plus de facteurs ainsi qu'un grand nombre de nœuds. Cependant, la fidélité à la réalité est limitée devant la masse de paramètres à intégrer. De plus, pour prendre en compte un nouveau facteur, il est nécessaire de le modéliser, avec, là encore, un risque d'imperfection vis-à-vis de la réalité. En revanche, la simulation permet une très grande reproductibilité du scénario simulé ; le cumul des résultats et l'étude statistique de ce dernier peut potentiellement indiquer des bornes, des limites, etc. qu'il faut considérer avec parcimonie puisque le nombre de simulations effectuées ne peut être infini.

2.1.3 Prototypage
Les approches par prototype (*testbed*) se veulent les plus complètes car naturellement exhaustives quant aux facteurs pris en compte dans la manipulation, l'expérimentation étant réelle. Cependant, les manipulations sont difficilement reproductibles ; de ce fait, l'interprétation des résultats est complexe.

2.1.4 Emulation
L'émulation de protocoles et de réseaux est une méthode intermédiaire entre la simulation et le prototypage car elle permet d'obtenir des résultats plus réels que ceux obtenus par la simulation et mieux maîtrisés, notamment sur l'aspect reproductibilité, que ceux obtenus par prototypage. Selon sa mise

en œuvre, l'émulation peut être considérée comme une simulation dans laquelle des paramètres réels sont introduits (comme par exemple, un médium réel sur des nœuds simulés) ou, au contraire, comme une virtualisation du matériel pour une exécution dans un contexte pleinement maîtrisé (comme par exemple, des machines virtuelles s'échangeant des messages via un canal modélisé). L'utilisation de cette approche est judicieuse dans le cas où les expérimentations ne peuvent pas être réalisées à cause de l'absence d'équipements réels, par exemple lorsque les produits ne sont pas encore disponibles sur le marché, ou lors d'un passage à l'échelle impliquant un très grand nombre de nœuds alors impossible à déployer et trop onéreux. Dans ces cas là, une partie au moins des équipements peut-être remplacée par des modèles développés dans l'environnement virtuel.

2.2 Exemples de plateformes de tests couplant simulation, émulation et/ou prototypage

Sans être exhaustif – les solutions sont extrêmement nombreuses et diverses – une sélection de dispositifs utilisés dans l'ingénierie des protocoles est présentée dans les sections suivantes. On s'intéressera particulièrement aux simulateurs de réseaux, aux émulateurs et aux plateformes de prototypage, avant de présenter l'environnement que nous proposons.

2.2.1 Network Simulator (NS-2, NS-3)

Les logiciels de simulation réseau NS-2 et NS-3 sont des simulateurs à événements discrets, fruits du développement de nombreux travaux de recherche. Ils sont très utilisés et reconnus par la communauté scientifique.

Plus particulièrement, dans le cadre de nos travaux sur l'émulation et le prototypage rapide, nous notons que NS-3 offre les deux possibilités d'émulation évoquées plus haut :

- A l'aide d'un médium réel utilisé pour véhiculer les données de façon concrète, NS-3 peut mettre en œuvre des protocoles simulés,
- A l'aide de machines virtuelles, il lui est possible d'émuler des nœuds complets – c'est-à-dire intégrant le système d'exploitation, les multiples logiciels serveurs ou clients, etc. – s'échangeant des messages à travers un canal modélisé par le simulateur réseau intégrant des modèles de couche physique très fidèles.

Il existe de nombreux travaux visant à intégrer le médium sans fil réel dans la simulation de protocoles par NS-3. On citera à titre d'exemple la plateforme ORBIT [1] constituée de 400 nœuds IEEE 802.11 répartis sur une grille à deux dimensions. Dans ce cas, les protocoles sont, eux, simulés par NS-3 et donnent lieu à des échanges sur le médium réel.

Des travaux tels que [2] visent à apporter des solutions d'émulation de IEEE 802.15.4 sous NS-3.

2.2.2 OPNET

OPNET est un logiciel de simulation réseau utilisé par la communauté scientifique, mais également par les industriels des réseaux et télécoms. Contrairement à *Network Simulator*, il est maintenu (et commercialisé) par une entreprise ; son utilisation est généralement payante.

On retrouve, dans OPNET, des possibilités de connexion entre des modèles de communications développés dans le logiciel de simulation, et des nœuds réseaux appartenant à une plateforme de communication réelle. Ces derniers sont totalement physiques ; on parle alors de *co-simulation*. Nous pouvons citer par l'exemple les travaux de [3] qui a utilisé cette approche pour connecter le contrôleur d'un pendule inversé avec des capteurs et des actionneurs via un réseau de communication simulé dans le logiciel OPNET *Modeler*. Dans le monde filaire, on peut aussi citer les travaux de [4] qui a utilisé lui aussi la co-simulation sous OPNET pour l'évaluation des performances de systèmes de communications IP sur bus industriel dédié à la distribution électrique.

OPNET se base sur l'utilisation de deux modules essentiels connus sous le nom de « *Hardware In The Loop* » (HITL) et « *Software In The Loop* » (SITL). Le module HITL assure la communication entre les équipements réels existant hors de l'environnement de simulation et les modèles conçus dans le logiciel de simulation. Le module SITL permet de créer des scénarios de simulation. La Figure 1, extraite de[4], illustre la structure de la plateforme de co-simulation sous OPNET intégrant ces modules.

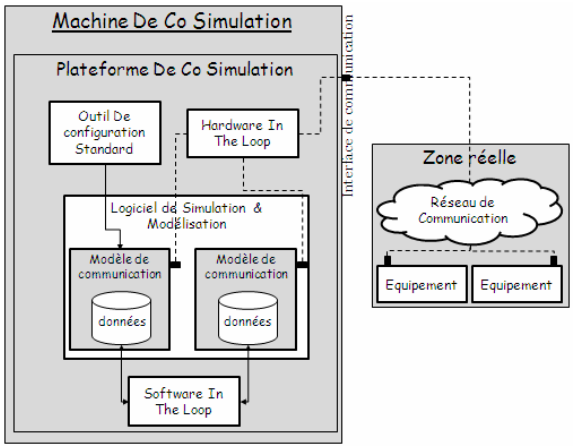

Figure 1 : Architectures de co-simulation sous OPNET[4]

Le module HITL assure la communication entre les modèles de communication conçus dans le logiciel de simulation et les équipements physiques qui appartiennent à une zone réelle. L'interface de communication de la machine de co-simulation permet à ces deux entités de communiquer. Le module SITL donne, à un outil externe au logiciel de simulation, un accès direct aux variables et au comportement des modèles de simulation. Ce module permet un accès à la simulation lors de son exécution.

Cette approche de co-simulation donne de bons résultats, en particulier pour des interfaces filaires standard telles que *Ethernet* avec TCP/IP. L'un des plus gros problèmes rencontrés est la lenteur relative de l'exécution des modèles simulés face à la rapidité effective des nœuds réels. Il est en effet nécessaire de faire interopérer, dans une co-simulation en temps réel, les deux types d'entités. Malheureusement, la simulation est généralement plus lente, et seuls des systèmes de communications à bas débit et temps de réaction importants peuvent effectivement être co-simulés. Ceci est d'autant plus vrai si l'on souhaite simuler sur une même machine un grand nombre de nœuds virtuels, ce qui est l'un des attraits de la co-simulation comme évoqué ci-dessus. Afin d'obtenir un comportement temporel équivalent à un vrai nœud, on est souvent obligé de se limiter à seuls quelques nœuds simulés (parfois un seul !) sur l'ordinateur hébergeant la plateforme de co-simulation OPNET. Il est également conseillé bien sûr de disposer

d'une machine très performante en termes de vitesse processeur et de taille mémoire RAM. Une autre piste est de répartir sur plusieurs machines les différents nœuds simulés. Il faut alors mettre en place plusieurs modules HITL permettant l'interface aussi bien avec le mode réel, qu'entre les groupes de nœuds simulés et répartis sur plusieurs machines. Ceci a déjà été testé sur une co-simulation d'un réseau *Ethernet*, mais reste confidentiel pour le monde sans fil des réseaux de capteurs.

2.2.3 WSNet

Comme *Network Simulator*, *WSNet* est un simulateur de réseau à événements discrets, libre, mais originellement dédié aux réseaux de capteurs sans fil. Couplé à *WSim* [5], il permet de simuler complètement le WSN, depuis les trafics jusqu'au médium. L'impact des protocoles, les effets du canal modélisé et les temps d'exécution sur les nœuds peuvent être évalués par simulation.

L'avantage de cette solution est la simulation complète du nœud, tant qu'il est basé sur le composant *Texas Instruments* MSP430. De multiples plateformes matérielles (*TelosB*, *WiSMote* [6]) et systèmes d'exploitations dédiés (*TinyOS*, *FreeRTOS*, *Contiki*) sont supportés. Sur un ordinateur standard, 10 à 15 nœuds peuvent être simulés simultanément, en fonction des trafics et des protocoles mis en jeu.

Au-delà des aspects de simulation pure, des plateformes telles que Senslab [7], équipées de nœuds à l'architecture compatible, peuvent servir de support d'expérimentation des protocoles implémentés. En fonction des choix fait en amont par le développeur, le portage de la simulation vers le prototype peut être plus ou moins simplifié.

2.3 Autres structures et plateformes

Bien entendu, les plateformes évoquées dans les sections précédentes ne sont pas les seules disponibles aujourd'hui. Il existe en effet un très grand nombre de solutions d'émulation, de co-simulation, etc. mais aussi des plateformes matérielles très diversifiées, tant en terme d'architecture *hardware* (*Atmel*, *Chipcon*, *Freescale*) que de cibles applicatives (*Imote*, *XBee*, *Arduino*). A ce titre, les capteurs implémentés sur les modules sont aussi un critère de choix pour la cible, là encore, en fonction de l'applicatif (température, accéléromètre, etc. voire même l'accès à un bus tel qu'USB, I^2C, 1-wire ou série). Sur des solutions basées sur *Arduino*, par exemple, le coût d'un nœud est relativement faible (<40EUR), ce qui permet d'envisager relativement simplement des déploiements d'une vingtaine de nœuds ; lorsque le passage à l'échelle n'est pas un objectif de l'étude, cette approche est très intéressante, car, par rapport aux *testbeds* de masse comme Senslab, des scenarii proches de l'application finale (mobilité, interaction avec les utilisateurs, embarquement du nœud sur une personne ou sur un véhicule, etc.) sont tout à fait possibles et potentiellement fidèles à la réalité du déploiement réel à venir.

Cependant, nous notons que la plupart des solutions disponibles proposent généralement une intégration globale de simulation et/ou d'émulation et/ou de prototypage pour des systèmes complets. En pratique, pour le développeur, le prototypage rapide de protocole de bas niveau (en particulier sans aucune modification de code entre l'émulateur et la cible finale) n'est pas toujours possible, notamment sur des cibles légères. En effet, si l'aspect modulaire apporte souplesse et versatilité, l'empilement protocolaire complet peut vite devenir imposant pour une plateforme très contrainte, notamment en termes de mémoire (typiquement 2 ou 4 ko de RAM) et de CPU (typiquement 8 bits à 8 MHz). La solution « WiNo », proposée dans la suite de cet article, permet de répondre à ces exigences, en s'affranchissant par exemple d'un système d'exploitation, aussi petit soit-il.

3. PRÉSENTATION DE WINO

3.1 Principes généraux et objectifs

WiNo est une plateforme ouverte prête à accueillir des protocoles de niveau *liaison*, *réseau* et plus, pour les réseaux de capteurs sans fil. WiNo a été développé de façon à offrir un accès de bas niveau pour un développeur exigeant qui souhaite non seulement maîtriser précisément les temps d'accès au médium, la mise en veille et le réveil des nœuds, mais aussi les temps CPU et la gestion de la mémoire – ressources généralement restreintes sur les cibles matérielles des réseaux de capteurs. Que ce soit dans le but de piloter des politiques drastiques d'économie d'énergie ou pour le respect de contraintes temps réel, une telle maîtrise de l'ensemble des composants du nœud est nécessaire ; WiNo est une plateforme candidate à l'accueil de protocoles à fortes contraintes temporelles visant plusieurs mois de fonctionnement avec deux piles AAA [8]. Le dispositif fournit un environnement incluant la gestion de la couche physique (typiquement IEEE 802.15.4, mais sans que ce soit une limite) et les outils nécessaires au développement d'une pile protocolaire complète. Le processus d'ingénierie des protocoles est simplifié par les possibilités conjointes d'émulation des nœuds et de déploiement sur cible finale, sans modification de code entre les deux, ce qui représente un avantage majeur.

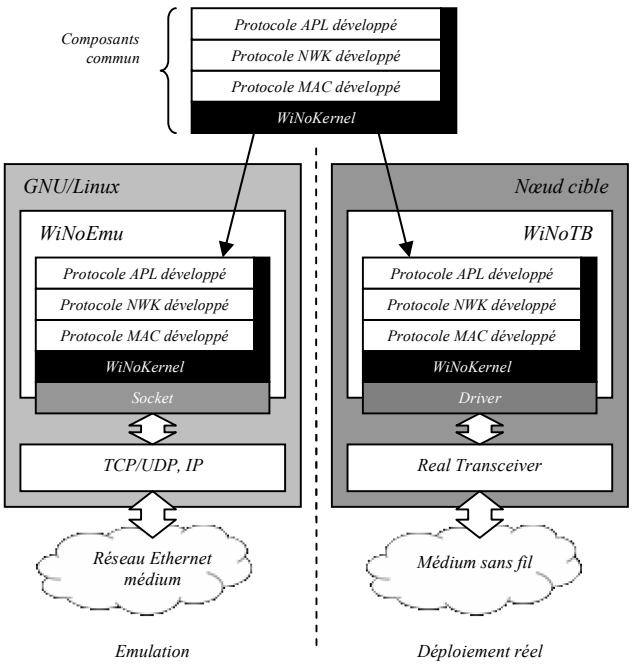

Figure 2 : structuration des composants logiciels de WiNoEmu (gauche) et WiNoTB (droite)

Pour arriver à ses fins et favoriser le prototypage rapide de protocoles, WiNo est constitué de deux sous-systèmes *WiNoEmu* (*Emulator*) et *WiNoTB* (*Testbed*) partageant un noyau commun *WiNoKernel* ; la Figure 2 illustre les liens entre les trois composants. L'usage typique de WiNo est le suivant : une fois le

noyau pris en main par le développeur, celui-ci peut commencer à implémenter le nouveau protocole et les algorithmes associés dans l'environnement WiNoEmu, sous GNU/Linux. Le langage utilisé est le C ; le développeur veillera à respecter constamment les contraintes imposées par la cible finale visée (mémoire limitée, éventuellement pas de variable de type flottant, etc.) bien qu'il dispose de plus de libertés s'il utilise l'émulateur. Pendant la phase d'implémentation, le développeur peut lancer plusieurs instances de WiNoEmu, chacune émulant un nœud du réseau et procéder ainsi à des tests. Une fois le nouveau protocole rodé sous l'émulateur, le développeur n'a plus qu'à intégrer le code à la cible finale grâce à WiNoTB. L'étape suivante est le déploiement et l'analyse de performance en environnement réel. En développant suivant les *méthodes agiles* [9], le processus de développement peut permettre d'obtenir rapidement des résultats dans le monde réel. Notons qu'en rapport avec le concept de co-simulation présenté au chapitre II, il n'est pour l'instant pas possible de mixer des nœuds émulés et des nœuds réels.

3.2 Détail des composants logiciels

3.2.1 Le noyau : WiNoKernel

WiNoKernel est le socle de WiNo. Il est commun à WiNoTB et WiNoEmu. Les cibles finales que nous visons étant *a priori* dépourvues de système d'exploitation, WiNoKernel fournit des outils variés, du *driver* de la couche physique à la gestion des couches protocolaires par machines d'états en passant par la gestion des tâches et des files d'attente. Les outils et mécanismes suivants sont pris en charge :

- Pilotage de la couche physique par un jeu complet de primitives adapté à un *transceiver* répondant aux spécifications de la PHY IEEE 802.15.4-2006,
- Gestion d'une horloge locale et mise à disposition d'une pile d'interruptions logicielles permettant des appels de fonctions avec exécution différée pour temporiser les actions sur le nœud,
- Séparation des niveaux OSI et gestion algorithmique par machines d'états dédiées à chaque niveau. Gestion d'une mémoire partagée permettant des échanges optionnels entre couches non adjacentes,
- Gestion des files d'attentes, des mécanismes d'encapsulation à l'émission, d'interprétation et d'étiquetage temporel (*timestamping*) à la réception,
- Au niveau 2 (MAC), gestion de l'adressage local et d'une table de voisinage permettant de gérer les acquittements et de détecter les doublons (LLC),
- Au niveau 3 (NWK), gestion du processus de routage.

WiNoKernel fournit l'ensemble des outils permettant une gestion simple et ouverte du nœud dépourvu d'OS. Les algorithmes de gestion étant ouverts et minimalistes, les aspects temporels peuvent être totalement caractérisés et maîtrisés pour satisfaire le développeur soucieux des aspects temps réel.

3.2.2 La plateforme d'émulation : WiNoEmu

WiNoEmu permet de tester les protocoles développés sur des nœuds émulés dans un environnement typique GNU/Linux. Grâce à WiNoEmu, le développeur peut procéder à des tests sans nécessité de manipuler l'environnement de *cross-compilation* et les cibles matérielles. Dans un cadre de collaboration recherche/enseignement, WiNoEmu peut permettre de sous-traiter l'implémentation du protocole à des étudiants développeurs qui n'ont pas nécessairement de compétences ni de pratique vis-à-vis du *hardware* spécifique.

WiNoEmu repose sur le noyau décrit ci-dessus et ajoute certaines fonctionnalités permettant d'émuler le médium sans fil : les notions de topologie, de taux erreur trame (FER, *Frame Error Rate*) et de dérive des horloges respectives des nœuds sont ainsi implémentées par WiNoEmu, ce qui permet d'atteindre un premier niveau de réalisme nécessaire pour la mise à l'épreuve d'un protocole en cours d'implémentation. La topologie est gérée par une matrice carrée dont la dimension est égale au nombre de nœuds. Chaque élément $\{i, j\}$ de la matrice est un réel compris dans l'intervalle [0,1]. Ce réel définit en premier lieu s'il y a connectivité ou non entre un couple de nœuds i, j : un 1 indiquera que le nœud j entend le nœud i, alors qu'un 0 indiquera une absence de connectivité, dans le même sens. Notons qu'en l'absence de lien radio asymétrique sur l'ensemble de la topologie, la matrice est symétrique. Notons également que les éléments de la diagonale principale sont nuls, puisqu'un nœud « ne s'entend pas lui-même », conformément aux spécifications des *transceivers* radio réels. Le FER est paramétré par cette même matrice : un élément $\{i, j\}$ non entier traduit ainsi la probabilité de bonne réception d'une trame émise par un nœud i et reçue par le nœud j. Comme c'est le cas sur l'ensemble des *transceivers* réels, une trame erronée (erreur détectée par le *Frame Check Sequence*, FCS) n'est pas remontée par la couche physique : elle est donc ignorée.

La Figure 3 donne un exemple de topologie d'un réseau de 8 nœuds, avec matrice de connectivité associée. Dans le cas représenté, le médium n'engendre pas d'erreur de transmission (le FER est nul, les éléments de la matrice sont des booléens).

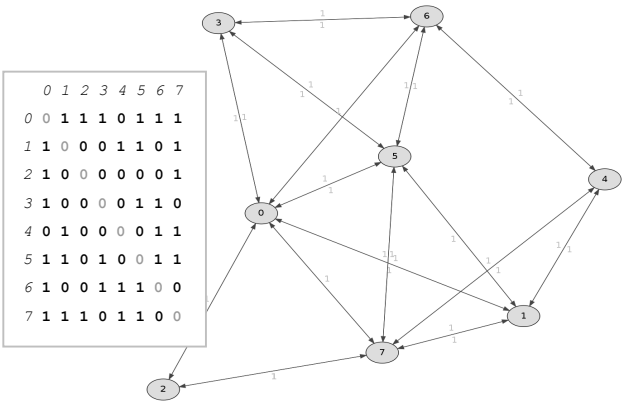

Figure 3 : Matrice de connectivité et topologie correspondante

L'émulation du médium proposée par WiNoEmu est simple et se veut efficace pour les tests du protocole en cours de développement ; il n'est pas question ici de se substituer à un simulateur de réseau en proposant une couche physique et un modèle de canal les plus fidèles possible (pas de modèle de propagation, d'antenne, etc.). Selon ce principe, les notions de *distance entre nœuds*, de *portée radio*, de *puissance d'émission* (*TxPower*) ou de *sensibilité de réception* ne sont pas décrites explicitement pour WiNoEmu. Seule la matrice de connectivité paramètre l'émulateur : dans cette approche de test du protocole avant portage sur cible réelle et déploiement, on considère assez simplement qu'une trame est reçue ou non par la couche Liaison, éventuellement avec un FER non nul. Dans le cas négatif, peu

importe s'il s'agit d'une erreur de transmission, d'une collision… ou d'une absence définitive de connectivité entre les deux nœuds. La véritable étude de performance sera ensuite réalisée sur le prototype avec WiNoTB.

WiNoEmu a deux modes de fonctionnement : client/serveur ou décentralisé. Dans un cas comme dans l'autre, les processus peuvent être exécutés par la même machine physique, ou sur *n* machines distinctes reliées en réseau.

En mode client/serveur, un processus serveur émule le médium et chaque nœud émulé est un processus client connecté au serveur. Dans ce mode de fonctionnement, toutes les communications se font en TCP et passent par le serveur qui, selon une probabilité appliquée à la matrice de connectivité décrite ci-dessus, renverra ou non les trames aux nœuds voisins émulés par les clients connectés au serveur. Dans ce mode de fonctionnement, les messages sont également estampillés par le serveur qui transmet cet horodatage aux nœuds destinataires ; en d'autres termes, c'est le serveur qui maintient (ou non) la synchronisation des nœuds. Selon la configuration de chaque nœud, le serveur peut également altérer l'estampille temporelle communiquée aux destinataires : cette altération permet d'émuler des décalages d'horloge entre nœuds du réseau. Enfin, toujours en fonction de l'estampille, le médium peut considérer comme perdues des trames émises à des instants suffisamment proches, en fonction de leur longueur : cas d'une collision.

En mode décentralisé, il y a *n* processus, pour *n* nœuds. Les messages sont transmis en diffusion par une *socket* UDP et chaque nœud partage la matrice de connectivité ; chaque message transmis est reçu par l'ensemble des nœuds qui déterminent eux-mêmes si le message est reçu ou perdu. Dans ce mode, les nœuds n'étant pas synchronisés, ni la désynchronisation, ni la détection de collisions ne sont disponibles. Notons que l'usage d'UDP ne garantit pas l'acheminement réel des trames émulées, contrairement au mode précédant.

Exécuté dans un environnement GNU/Linux, un réseau émulé avec WiNoEmu peut être analysé de différentes manières : via des scripts, par analyse *a posteriori* des *logs* retournés par la console de chaque nœud, ou bien en temps réel avec les nombreux outils rendus disponibles par la communauté *Open Source* : par exemple, une analyse en temps réel des messages échangés entre nœuds pourra être réalisée par les outils reposant sur la *libpcap*, comme *tcpdump*, *wireshark/ethereal*, etc.

Bien que fonctionnant sur un environnement type GNU/Linux, il est important que le développeur utilise WiNoEmu en intégrant les contraintes imposées par la cible finale, ceci dans le but d'en simplifier le portage du code avec WiNoTB.

3.2.3 La plateforme de prototypage rapide : WiNoTB

WiNoTB est l'équivalent de WiNoEmu mais sur cible réelle. Le même code est exécuté sur des nœuds réels, permettant déploiement et évaluation de performance en environnement réel. WiNoTB repose sur le même noyau que WiNoEmu et peut être utilisé directement, dès le début de l'implémentation. Passer de WiNoEmu à WiNoTB – et vice versa – est tout à fait possible et recommandé.

Actuellement, les plateformes supportées sont celles basées ou dérivées du couple *Freescale* microprocesseur *9S08GT60* et *transceiver MC1319x* : *13192-SARD*, *1321x-SRB*, *1321x-NCB* ou bien encore *Freescale ZRD01*. WiNoTB repose sur le *driver Open Source SMAC* [10] de *Freescale*, que nous avons optimisé, notamment sur l'utilisation des *timers* du composant radio. WiNoTB est en cours de portage sur *Arduino*, au-dessus de la bibliothèque *VirtualWire*.

Une fois le protocole porté sur la cible finale et le déploiement effectué, l'analyse de performance repose sur les outils communément utilisés tels que les analyseurs de protocoles. Cependant, grâce à l'architecture ouverte du nœud déployé, des analyses plus fines peuvent être menées, par exemple par la remontés des consoles sur un serveur central (*dump* des tables de voisinage ou de routage, *timestamping* des messages, etc.) grâce à un module Ethernet/USB pour ne pas biaiser l'étude des performances en chargeant le réseau à caractériser. La mesure fine et l'optimisation des temps d'exécution CPU peuvent également être réalisées directement sur cible finale comme illustré sur la Figure 4. A ce stade, l'architecture ouverte du nœud permet d'imaginer l'usage de nombreux outils pour évaluer les performances du protocole en conditions réelles.

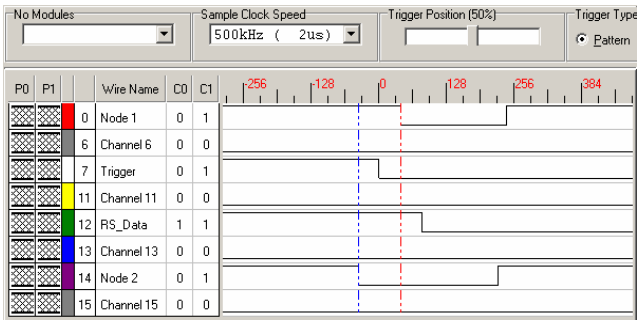

Figure 4 : évaluation des temps d'exécution CPU sur cible

4. EXEMPLES D'IMPLÉMENTATIONS SOUS WINO

Dans cette dernière section, nous détaillons quelques exemples de résultats obtenus avec WiNo dans le cadre de propositions de protocoles originaux.

4.1 SiSP

SiSP (*Simple Synchronization Protocol*) [11] est un protocole de synchronisation léger et réparti pour les réseaux de capteurs sans fil, sans hiérarchie. Il permet, en quelques itérations, d'obtenir par *consensus* une *horloge partagée*. SiSP a été implémenté sous WiNoTB et a été testé en conditions réelles sur un réseau de plusieurs nœuds.

La Figure 5 représente le consensus de synchronisation obtenu dans trois situations différentes : quatre nœuds tous à portée (*full-mesh*) (a), deux nœuds mobiles en approche (b), un lien asymétrique (c). L'axe des ordonnées représente la valeur de l'horloge pour les nœuds (chaque courbe représente l'horloge partagée vue d'un nœud), l'axe des abscisses symbolise le temps. L'objectif de SiSP étant que les nœuds convergent vers une même horloge partagée, le consensus est obtenu lorsque les points sont superposés. Les résultats présentés ici ont été obtenus par l'usage d'un analyseur de protocole [12].

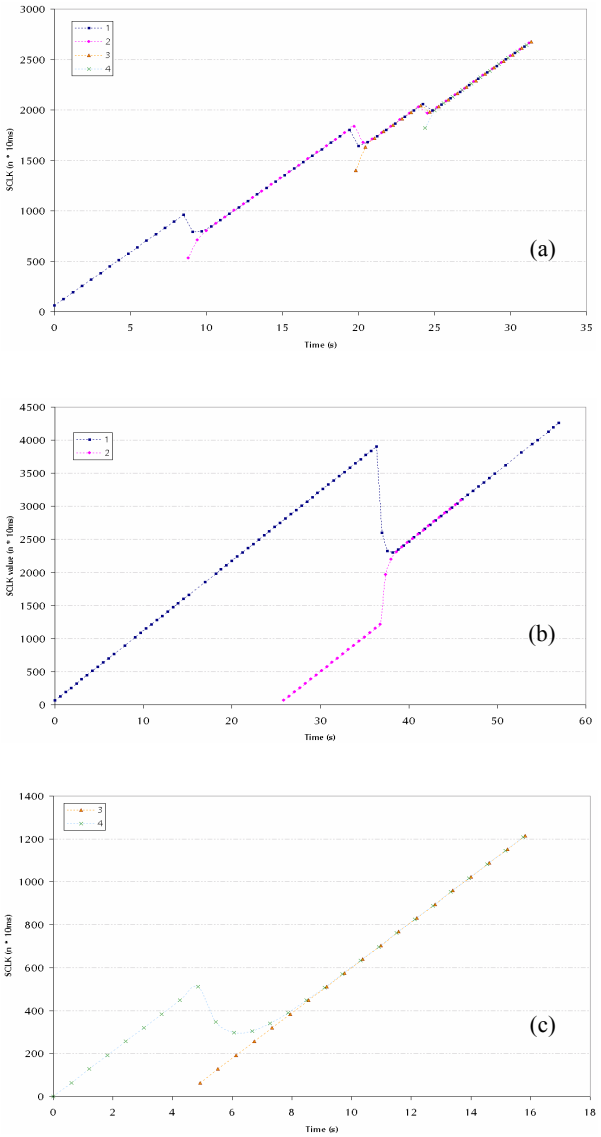

Figure 5 : Résultats obtenus par l'implémentation de SiSP sous WiNo

4.2 ADCF

ADCF (*Adaptive and Distributed Collision-Free*) [13] est un protocole MAC original permettant l'auto-organisation d'un ensemble de nœuds IEEE 802.15.4 sur une topologie sans hiérarchie. ADCF repose sur l'approche *Beacon Only Period* (BOP) pour synchroniser les activités des nœuds du réseau. Comme toujours, en dehors des périodes actives, les nœuds somnolent et économisent de l'énergie.

La première contribution amenée par ACDF porte sur l'organisation de la BOP qui est réalisée de façon distribuée, considérant une topologie maillée. Les nœuds s'accordent alors des *Collision Free Beacon Slots* (CFBS) en tenant compte de leur vision à deux sauts, pour éviter les collisions. La notion de *PAN-Coordinator* est alors abrogée et le réseau fonctionne sans aucune hiérarchie. L'absence d'arbre ou de *cluster* dans la topologie permet une gestion plus flexible en cas d'ajout, de suppression ou de mobilité d'un nœud. La seconde contribution d'ADCF concerne la distribution des slots garantis entre nœuds. En effet, en l'absence de nœud coordinateur, le concept de *Guaranted Time Slot* (GTS) de 802.15.4 n'est plus utilisable. De plus, là encore, considérant l'environnement maillé, il est nécessaire d'avoir une vision à deux sauts pour garantir l'exclusivité de l'accès au médium. ADCF propose des *Collision Free Data Slots* (CFDS) négociés entre paires de nœuds pour s'échanger des données dans un mode sans contention.

Après une étude en simulation réalisée sous OPNET, ADCF a été implémenté sous WiNo, d'abord sur WiNoSimu, puis sur WiNoTB. Un réseau réel constitué de plusieurs nœuds hétérogènes est actuellement déployé et l'état du réseau est visible en temps réel [14]. Dans le cadre de la validation d'ADCF, après l'étude par simulation, le déploiement a permis d'évaluer le protocole en situation réelle. Outre des tests pragmatiques comme le temps d'acheminement d'une trame ou le débit constaté par l'utilisateur, la mise en veille des nœuds étant implémentée sur WiNoTB, une étude sur la durée de vie des nœuds alimentés par batteries est actuellement en cours.

4.3 SiRP

SiRP (*Simple Routing Protocol*) est une implémentation d'un protocole de routage réactif qui repose sur des principes proches d'AODV (recherche d'une route à la demande, par inondation de messages de type *RouteRequest* et retour d'un ou plusieurs messages de type *RouteResponse*).

Dans un premier temps, SiRP a été implémenté pour tester les fonctionnalités « routage » de WiNo (files d'attente du niveau 3, table de routage, détection de l'absence de route et déclanchement du processus de recherche, etc.). Dans un second temps, il a été utilisé dans le cadre des travaux liés à ADCF présentés ci-dessus.

5. CONCLUSION ET PERSPECTIVES

Dans cet article, a été présenté WiNo, une plateforme d'émulation et de prototypage rapide pour l'ingénierie des protocoles destinés spécifiquement aux réseaux de capteurs sans fil. Cette solution se compose de deux éléments : d'une part, un émulateur, fonctionnant en environnement type GNU/Linux, qui permet le développement et les tests rapides du protocole avec un minimum de paramétrage (une simple matrice de connectivité) ; d'autre part, un environnement de prototypage (*testbed*) qui permet le test du protocole en environnement réel, sur cible finale. Les deux éléments partagent le même code et reposent sur un noyau commun qui simplifie le passage de l'un à l'autre, et inversement. Tout en proposant un accès fin et de bas niveau nécessaire à l'implémentation de tout protocole MAC ambitieux et respectueux sur des contraintes temporels et énergétiques, il permet clairement d'accélérer le processus d'implémentation et d'obtenir rapidement une maquette opérationnelle. Le dispositif répond aux attentes et a permis, en une année d'existence, le maquettage de trois protocoles originaux dont certains résultats ont été présentés dans cet article.

Les perspectives sont nombreuses : dans un premier temps, l'élaboration d'une IHM pour l'émulateur est en cours de finalisation ; elle permet de lancer simplement de nombreuses instances de WiNoEmu et de récupérer les résultats obtenus par analyse des fichiers de *logs*. De plus, le portage du noyau sur d'autres plateformes matérielles est également prévu : un portage sur *Arduino*, via les librairies *VirtualWire* et *RF22* d'une part, permettra de profiter de la modularité de ces plateformes et de la grande communauté de développeurs et utilisateurs. D'autre part,

il est planifié le développement d'un *driver* permettant de piloter les nouvelles couches PHY de la norme IEEE 802.15.4a, ouvrant de nouvelles perspectives pour les concepteurs de protocoles, en particulier sur la localisation de stations mobiles communicantes, au cœur également de nos travaux de recherche.

6. REMERCIEMENTS

Les auteurs tiennent à remercier Vincent Bragard, Damien Clerc et Rémy Phelipot pour leur aide sur le développement et la validation de WiNo.

7. REFERENCES

[1] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu and M. Singh, Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols, *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2005)*

[2] R. Beuran, J. Nakata, T. Yasuo, Y. Shinoda, IEEE 802.15.4 Network Emulation Testbed, *IEEE International Conference on Advanced Information Networking and Applications (AINA), 22-25 mars 2011, pp 451 - 458*.

[3] M-S. Hasan, H. Yu, A. Griffiths, T-C. Yang, Co-simulation framework for Networked Control Systems over multi-hop mobile ad-hoc networks, *17th World Congress, The International Federation of Automatic Control', IFAC 2008, pp. 12552 – 12557*

[4] M. Haffar, Modélisation et évaluation des performances et de la sûreté de fonctionnement de chaînes de contrôle/commande des installations de distribution électriques, *thèse de doctorat, Gipsa-Lab, Grenoble, 2011*

[5] G. Chelius, A. Fraboulet, E. Fleury, Worldsens: a fast and accurate development framework for sensor network applications, *22ième Symposium ACM « Applied Computing » (SAC 2007) (Séoul, Corée du sud), mars 2007*.

[6] WiSMote : http://wismote.org

[7] Senslab : http://www.senslab.info

[8] N. Fourty, A.van den Bossche, T. Val, An advanced study of energy consumption in an IEEE 802.15.4 based network: everything but the truth on 802.15.4 node lifetime, *Computer Communications, Elsevier, Numéro spécial Wireless Green, juin 2012*

[9] Agile Alliance http://www.agilealliance.org

[10] G. Delgado Huitrón, J. Herrero Gallardo, SMAC - Wireless connectivity made easy. http://www.freescale.com/files/microcontrollers/doc/support_info/BeyondBits2article16.pdf

[11] A. van den Bossche, T. Val, R. Dalce. SISP: a lightweight Synchronization Protocol for Wireless Sensor Networks. *Emerging Technologies and Factory Automation (ETFA 2011), Toulouse, 05/09/2011-09/09/2011, IEEE Computer Society*

[12] The Daintree network analyzer http://www.daintree.net

[13] J. Lu, A. van den Bossche, E. Campo, An Adaptive and Distributed Collision-Free MAC Protocol for Wireless Personal Area Networks. *International Symposium on Intelligent Systems Techniques for Ad hoc and Wireless Sensor Networks (IST-AWSN 2011), Niagara Falls - Canada, 19/09/2011-21/09/2011, Elsevier Science*

[14] ADCF Live demonstration http://lab.iut-blagnac.fr/adcf