Évaluation et amélioration des plates-formes logicielles pour réseaux de capteurs sans-fil, pour optimiser la qualité de service et l'énergie

Kévin Roussel

INRIA Nancy Grand-Est — LORIA UMR 7503 — Université de Lorraine

Directeurs de thèse : Ye-Qiong SONG et Olivier ZENDRA

3 juin 2016

Introduction — sommaire

Introduction I

Réseaux de capteurs sans-fil (WSN)

- Constitués de nœuds nommés **capteurs sans-fil** ou *"motes"*
- Interconnexion entre eux → ***Internet of Things (IoT)***

Plates-formes logicielles (OS) spécialisées dans les WSN

- ... avec **piles protocolaires réseau** spécialisées
- ***Maillon faible = couches basses de ces piles réseau***
  (comme nous allons le démontrer)

Introduction II

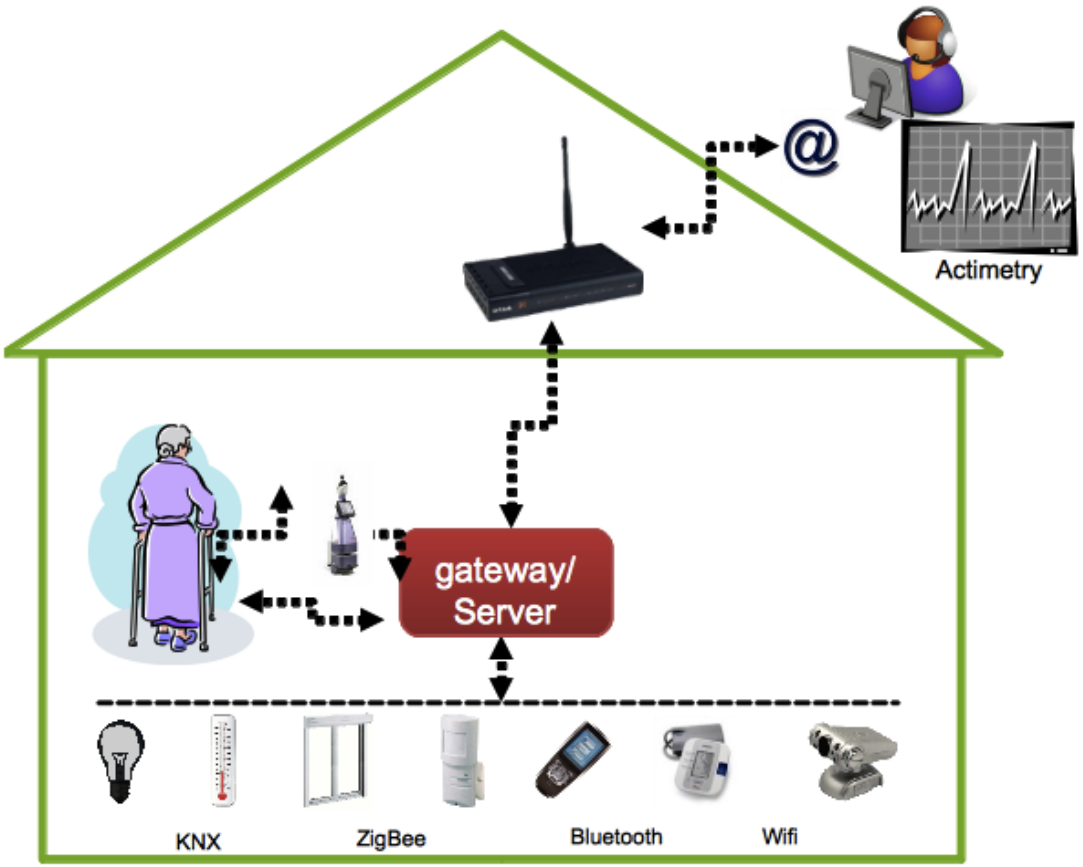
But de la thèse :

Développer des méthodes pour améliorer ces couches basses, en exploitant toutes les fonctionnalités offertes par ces OS dédiés

Contexte et problématique — sommaire

Contexte : applications d'e-santé des WSN

Un domaine déjà bien établi et très actif



Le projet LAR : *"Living Assistant Robot"*

But : aider au maintien à domicile des personnes âgées / dépendantes, en retardant au maximum leur placement en institutions spécialisées.
→ Financement de la présente thèse.

Problématique de la thèse

Couches des piles protocolaires réseau dédiées aux WSN

Deux extrêmités dans les piles réseau pour WSN / IoT :

- Les couches hautes (routage, applications, Web des objets, etc.)
  $\rightarrow$ nombreux travaux réalisés et publiés
- Les couches basses (physique / pilotes radio, MAC / RDC [1])
  $\rightarrow$ pas d'implémentations ***à la fois*** efficaces, portables et standardisées

Faiblesses des couches basses :

- Sapent les fondations des travaux sur couches hautes
- ***« Château bâti sur du sable »***

1. **RDC :** *Radio Duty Cycle* (Cycle de fonctionnement — activation et désactivation — de l'émetteur / récepteur radio)

Postulats de notre thèse

Notre Thèse

Une fondation solide pour l'IoT passe par le développement de :

- ***des protocoles MAC / RDC avancés à haute QdS*** [1]***,***
- ***sur un OS dédié WSN :***
  - ***reconnu et répandu,***
  - ***offrant les fonctionnalités requises (temps-réel).***

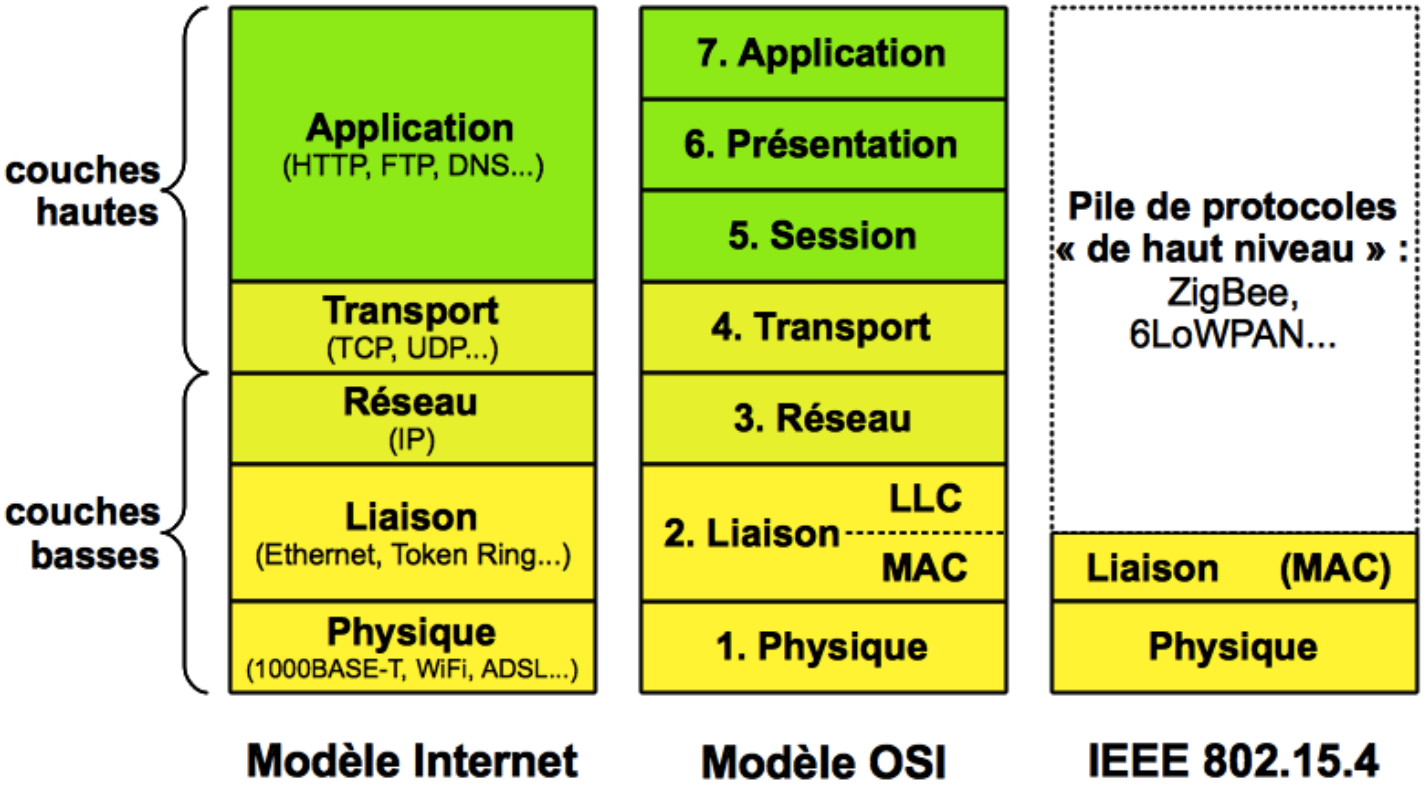
1. **QdS :** qualité de service *(QoS :Quality of Service)*

Sommaire général

Analyse critique de l'état de l'art — sommaire

Le protocole IEEE 802.15.4

Protocole 802.15.4 : ne concerne que les deux couches les plus basses (1 — PHY & 2 — MAC)



Protocoles MAC I

Difficulté : synchronisation (points de rendez-vous) entre nœuds avec des *duty cycles* différents

Nombreux protocoles MAC alternatifs

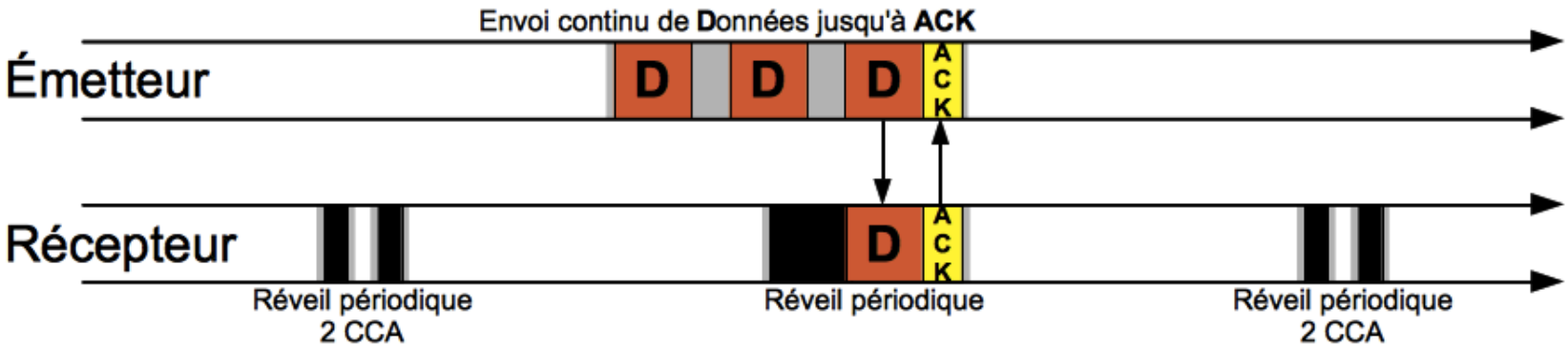
Basés sur différentes approches :

- Contention (CSMA/CA)[1] :
  - Synchrones
  - Asynchrones :
    - Écoute à basse énergie (LPL : *Low-Power Listening*)
      $\rightarrow$ SI : *Sender-Initiated*
    - Émission à basse énergie (LPP : *Low-Power Probing*)
      $\rightarrow$ RI : *Receiver-Initiated*
- Multiplexage :
  - Temporel (TDMA : *Time Division Multiple Access*)
  - Fréquentiel (FDMA : *Frequency Division Multiple Access*)
- Hybrides

1. **CSMA/CA :** *Carrier Sense Multiple Access with Collision Avoidance* (Écoute d'un Support à Accès Multiple, avec Évitement des Collisions)

Protocoles MAC II

Exemple de protocole MAC / RDC classique et très utilisé : **ContikiMAC**

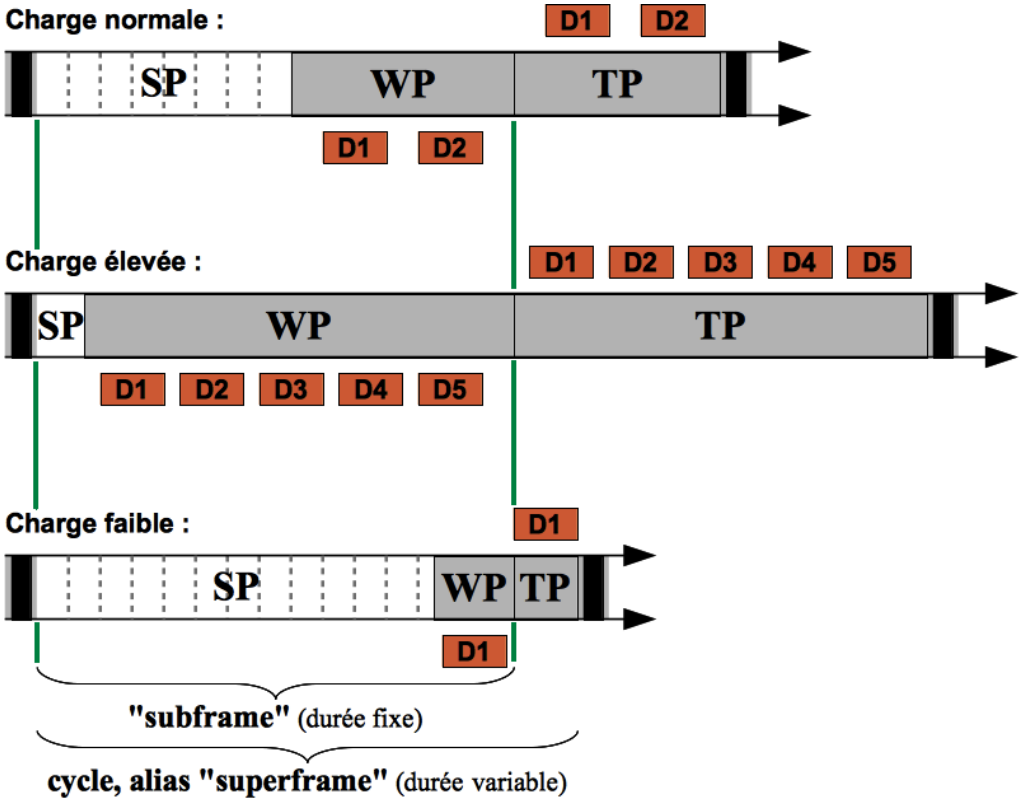


- Basé sur la contention (CSMA / CA, paradigme LPL)
- Cycle de fonctionnement fixe (défini à la compilation), correspondant au délai entre deux écoutes successives du médium radio
  $\rightarrow$ Adaptation « réactive » au trafic (écoute tant que médium occupé)
  + optimisation récente : *"phase lock"*

Protocoles MAC III

Exemple de protocole MAC / RDC avancé : **S-CoSenS**

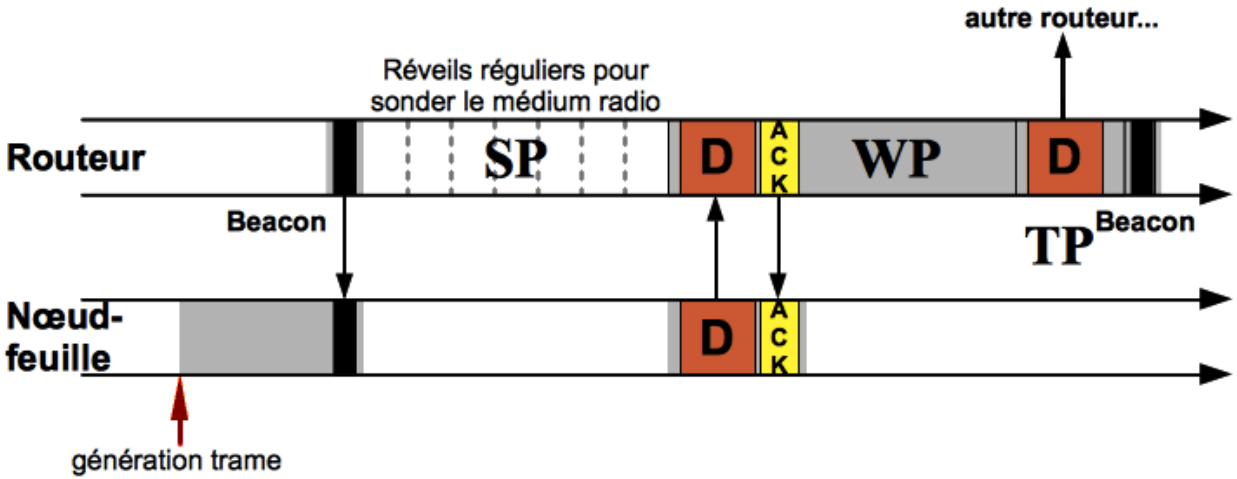
- Adaptation ***calculée*** au trafic radio :



Protocoles MAC IV

Exemple de protocole MAC / RDC avancé : **S-CoSenS** (suite et fin)

- Contention, hybride LPP / LPL :
  - LPP (RI) → communication intra-PAN [1]
  - LPL → communication entre routeurs (inter-PAN)
- Transmission d'un paquet :



1. **PAN :** *Personal Area Network* (Réseau d'Étendue Personnelle ; réseau élémentaire dans le domaine des WSN)

Protocoles MAC V

Synthèse sur les protocoles MAC / RDC

Différence fondamentale entre protocoles « classiques » et « avancés »

Adaptation au trafic réseau :

- protocoles « classiques » (802.15.4[e], ContikiMAC...) : ***configuration statique ou adaptation réactive par essai / erreur*** (allongement de la période active en cas de trafic pour ContikiMAC)
- protocoles « avancés » (S-CoSenS, iQueueMAC...) : ***adaptation proactive par calcul des besoins*** au trafic réseau (par mesure ou estimation de l'intensité de ce trafic)

En résumé :

« classique » = réactif / « avancé » = proactif

Systèmes d'exploitation spécialisés I

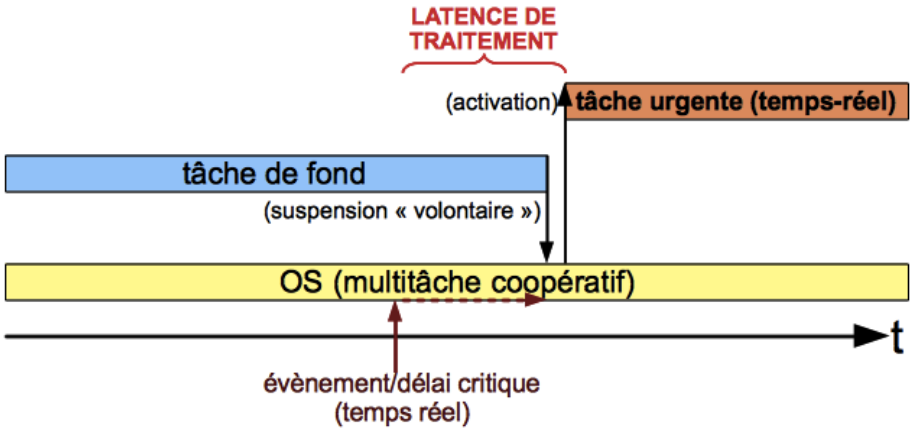
Multiples OS développés pour les WSN

Différences :

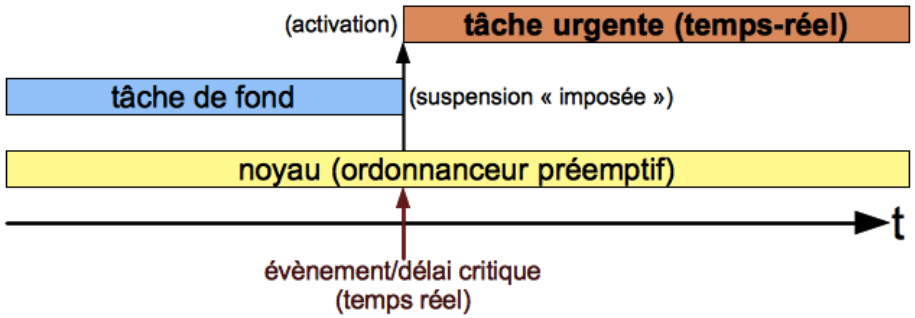
- fonctionnalités offertes
- exigences matérielles
- diffusion / adoption

Systèmes d'exploitation spécialisés II

Rappel sur les notions de multitâche et de temps-réel :



Modèle multitâche **coopératif** et temps-réel



Modèle multitâche **préemptif** et temps-réel

Systèmes d'exploitation spécialisés III

OS	Capacités	Nous recommandons pour les protocoles...
TinyOS	MT coopératif	asynchrones
Contiki	MT coopératif, temps-réel « basique »	basés sur la contention
LiteOS	MT préemptif	basés sur la contention
NanoRK	MT préemptif, temps-réel	tous : contention, multiplexage, avancés
RIOT OS	MT préemptif, temps-réel	tous : contention, multiplexage, avancés
FreeRTOS	*Noyau* MT préemptif, temps-réel	tous : contention, multiplexage, avancés (OpenWSN)

Systèmes d'exploitation spécialisés IV

Synthèse sur les OS pour WSN : choix de PF logicielles

Plates-formes logicielles choisies :

- **Contiki OS** : référence actuelle dans le domaine des WSN — standard de fait, au moins dans la recherche académique
- **RIOT OS** : plate-forme très récente, performante — notamment sur les fonctionnalités temps-réel —, et au développement dynamique et ouvert (auquel nous avons participé)

Plates-formes logicielles : évaluation, problèmes et améliorations — sommaire

Contiki OS I

Contiki OS : avantages

- Standard de fait actuel : très répandu et utilisé
- Très peu gourmand en ressources matérielles
- Nombreuses fonctionnalités réseau : ContikiMAC, uIPv6, Rime...
- Nombreux portages (architectures MCU [1] et matériels)
- Codé en C (avec quelques limitations)
- Fonctionne de base avec **Cooja** : simulateur / émulateur de WSN

1. **MCU :** *MicroController Unit* (Microcontrôleur)

Contiki OS II

Contiki OS : limites bloquantes

- Multitâche coopératif : *"protothreads"*
- Documentation minimaliste
- Limites techniques :
  - pilotes radio incomplets durant nos travaux [1]
  - pile réseau centrée sur un unique *"`packetbuf`"*
  - complexité excessive de la pile réseau
- Fonctionnalités temps-réel insuffisantes (`rtimer`) :
  - une seule instance de `rtimer`
  - code du système non réentrant
  - sans `rtimer` : granularité très insuffisante (8 ms)
- Coopération avec l'équipe de développement quasi-impossible

1. (corrigé dans la *release* 3.0)

RIOT OS I

RIOT OS : avantages

- Micro-noyau à multitâche préemptif
- Gestion avancée du temps-réel (avec fine granularité $\rightsquigarrow$ 1 μs)
- Soin apporté à la qualité du code
- Codé en C standard (ISO C 99) sans limitations
- Architecture modulaire : tout est optionnel hors micro-noyau
- Équipe de développement ouverte et réactive

RIOT OS II

RIOT OS : inconvénients

- Projet récent : premières versions publiées en 2013
- Porté sur moins d'architectures MCU que, par exemple, Contiki :
  - ARM (v7 puis Cortex-M)
  - MSP430
  - AVR
- Développement rapide, mais encore « jeune » sur certains points :
  - API des *timers* refaite : `xtimer`
  - Nouvelle pile protocolaire réseau : `gnrc`

  $\rightarrow$ nécessité de recoder les applications

RIOT OS III

Contributions techniques (*Pull Requests* sur GitHub)

- Gestion des erreurs fatales (`core_panic()` ajoutée au noyau)
- Portage de RIOT sur plate-forme matérielle Zolertia Z1
- Déboguage de RIOT OS sur architecture MCU MSP430
- Autres contributions (déboguage et améliorations $\rightarrow$ $\approx$ une trentaine)

Autre contribution

Analyse critique de la nouvelle pile réseau (`gnrc`) de RIOT OS

Évaluation et comparaison de protocoles MAC / RDC — sommaire

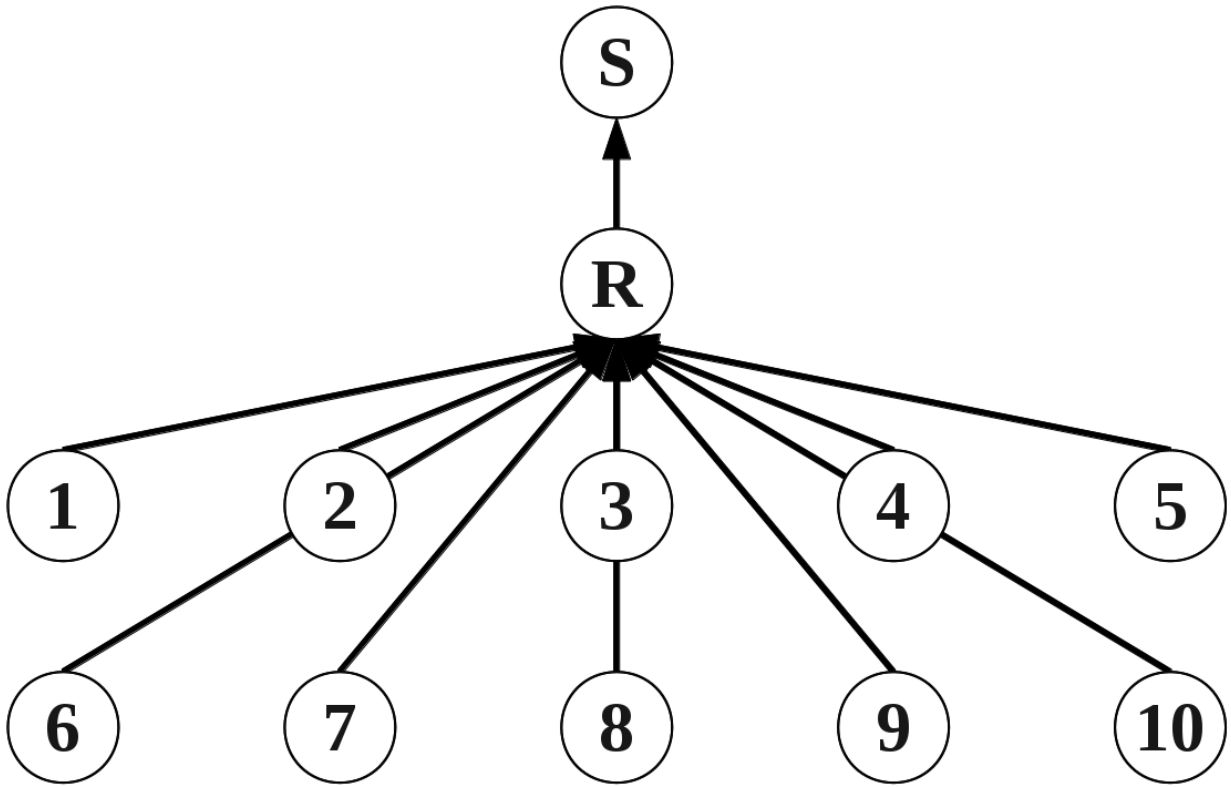
Premières plates-formes matérielles
TelosB / SkyMote et Zolertia Z1

Premières plates-formes matérielles (surtout émulées)

- Basées sur l'architecture MSP430 :
  - MCU MSP430F1611 pour la TelosB / SkyMote (8 MHz, 48 Ko Flash, 10 Ko RAM)
  - MCU MSP430F2617 pour la Zolertia Z1 (16 MHz, 92 Ko Flash, 8 Ko RAM)
- Même émetteur /récepteur radio : TI ChipCon CC2420
- Divers capteurs physiques (plus bus d'extension pour capteurs supplémentaires — *"Phidgets"* — sur Zolertia Z1)
- En commun : antenne intégrée, port pour antenne externe, mémoire Flash hors MCU (stockage permanent de données)

Implantation de S-CoSenS sous RIOT OS I

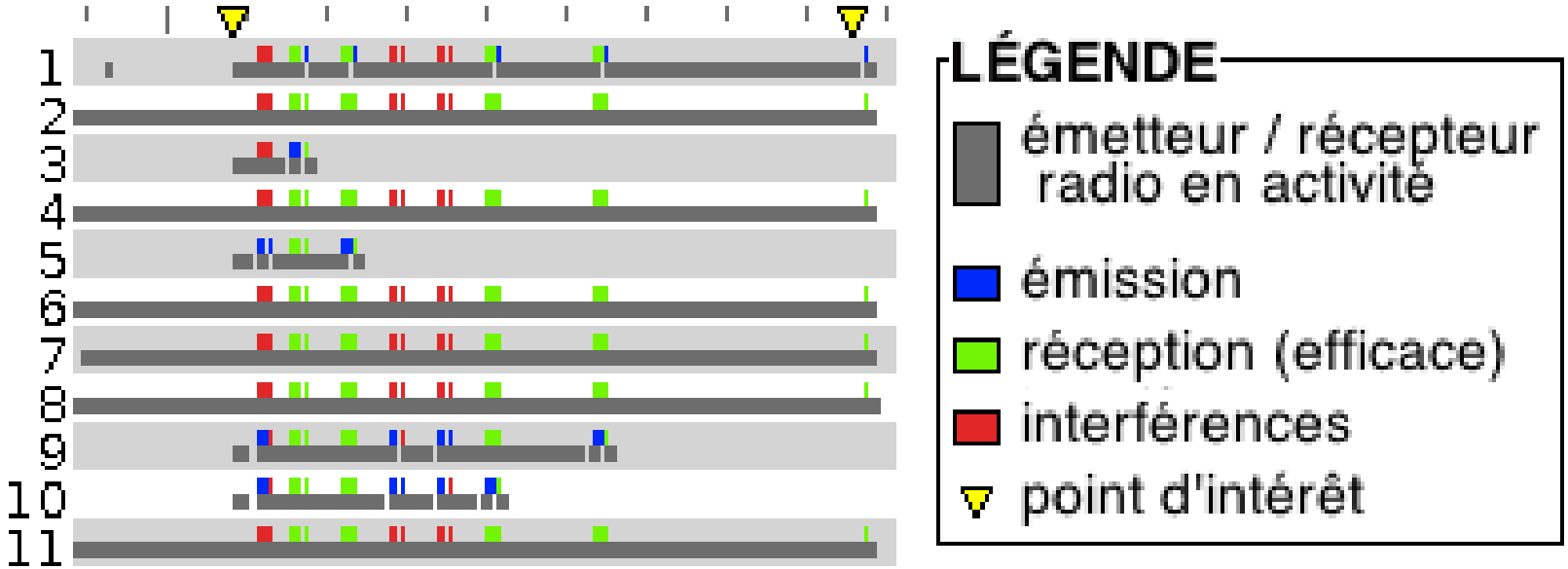
PAN virtual de test (schéma structurel) :



Implantation de S-CoSenS sous RIOT OS II

Premier test sous Cooja

Synchronisation précise et efficace entre nœuds, grâce aux fonctionnalités temps-réel de RIOT OS :



Comparaison avec ContikiMAC I

Évaluation : configuration des expériences

Débit réseaux utiles programmés sur l'ensemble des dix nœuds-feuilles :

Configuration	**PTI [1] moyen (par nœud)**	**Fréquence TX moyenne totale**	**Débit total attendu**
Modérée	1500 ms	6,7 trames / s	5867 bit / s
Élevée	1000 ms	10 trames / s	8800 bit / s
Très élevée	500 ms	20 trames / s	17600 bit / s
Extrême	100 ms	100 trames / s	88000 bit / s

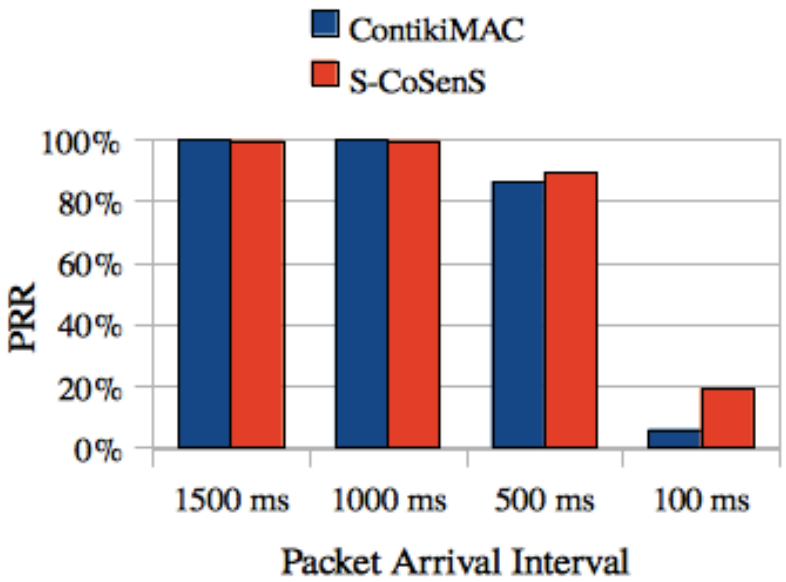
File d'envoi d'une taille de 8 paquets pour tous les tests.

1. **PTI :** *Packet Transmission Interval*, Intervalle d'Émission des Paquets / trames

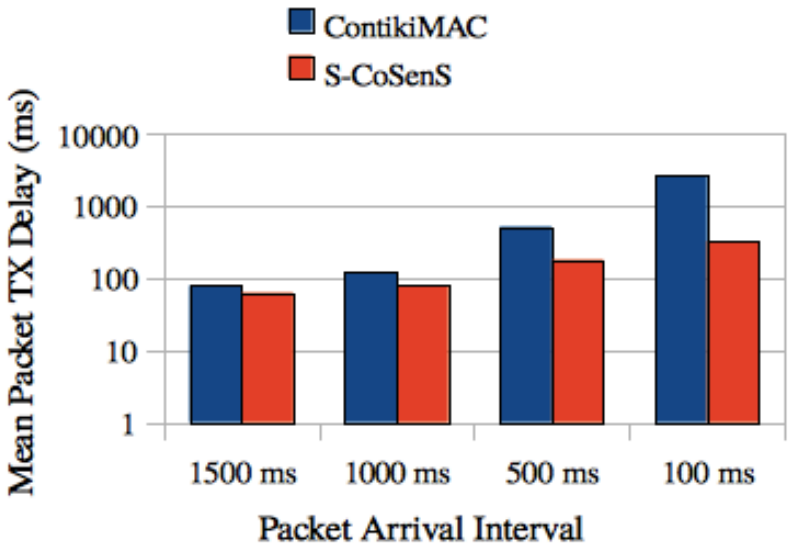
Comparaison avec ContikiMAC II

Qualité de service

S-CoSenS / RIOT aussi bon ou meilleur que ContikiMAC / Contiki OS en termes de Qualité de Service (**QdS**) :



Taux de réception de paquets (**TRP** / *PRR*)



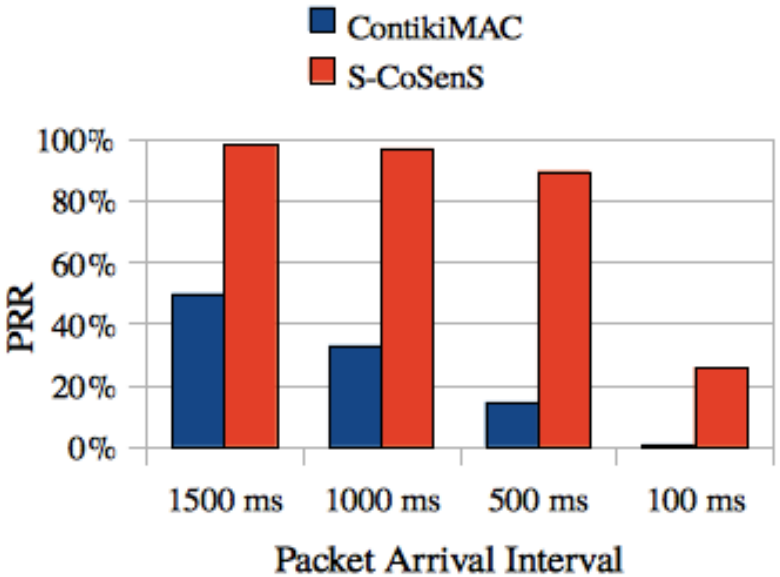
Délais de transmission *end-to-end*

Note : cycles de 32 Hz / 31 ms utilisés pour avoir des résultats comparables en termes de TRP entre S-CoSenS et ContikiMAC

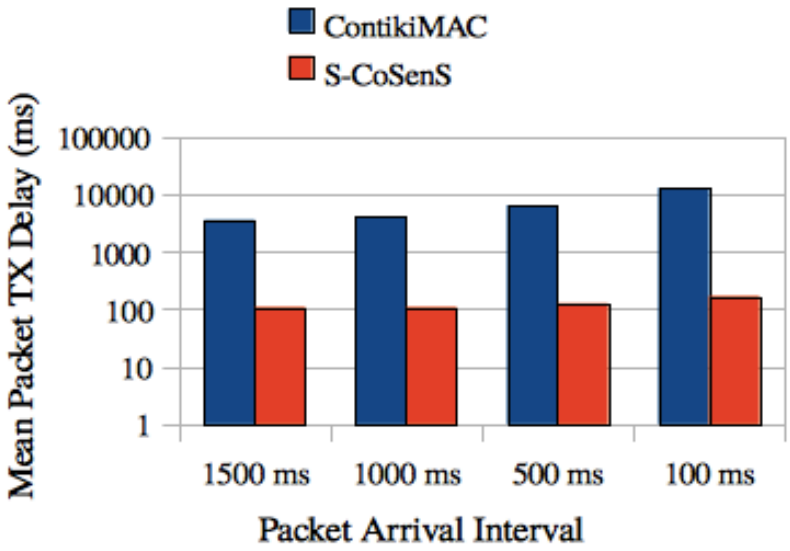
Comparaison avec ContikiMAC III

Qualité de service (bis)

S-CoSenS / RIOT largement meilleur que ContikiMAC / Contiki OS en termes de Qualité de Service (**QdS**) avec des cycles longs :



Taux de réception de paquets (**TRP** / *PRR*)



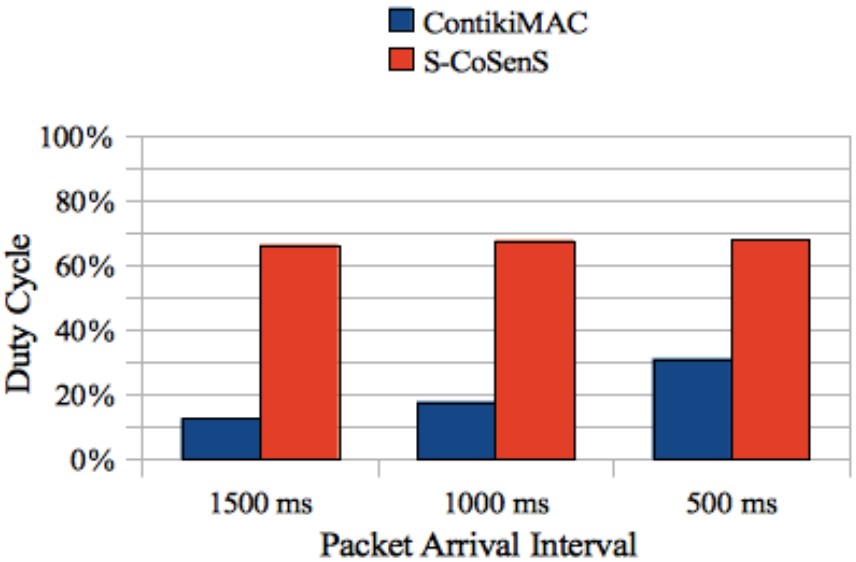
Délais de transmission *end-to-end*

Note : cycles de 8 Hz / 125 ms (durée de cycle par défaut pour ContikiMAC)

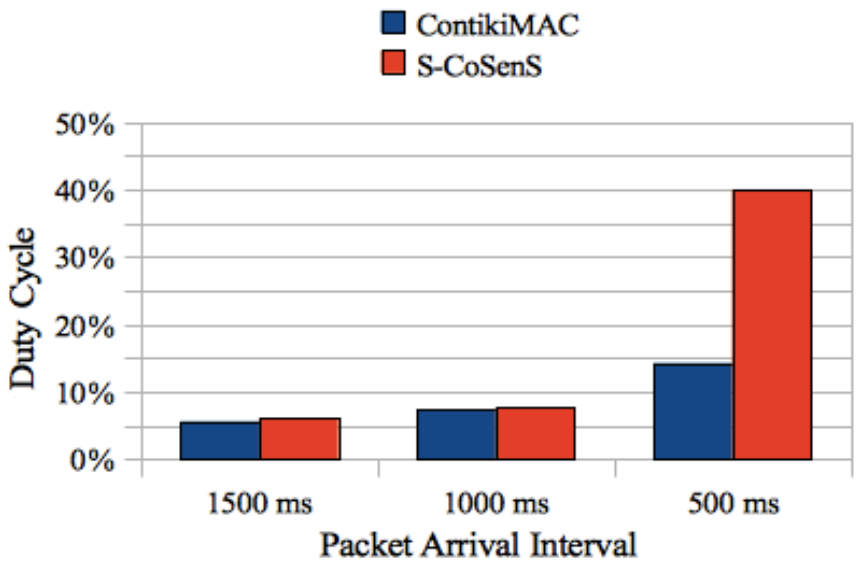
Comparaison avec ContikiMAC IV

Consommation d'énergie (approximation par les *duty cycles*)

ContikiMAC / Contiki OS meilleur que S-CoSenS / RIOT en termes de *duty cycles* :



Duty cycles estimés du nœud routeur (activité radio totale)



Duty cycles estimés des nœuds feuilles (activité radio totale)

Note : cycles de 32 Hz / 31 ms utilisés pour avoir des résultats comparables en termes de TRP entre S-CoSenS et ContikiMAC

Comparaison avec ContikiMAC V

Discussion

Constatations issues de ces comparaisons par simulation

- S-CoSenS / RIOT OS meilleur concernant la Qualité de Service
- ContikiMAC / Contiki OS meilleur concernant les *"duty cycles"*
- Problèmes de stabilité : débordements mémoire
  $\rightarrow$ MPU, ou code de contrôle ajouté au processus de compilation
- Influence de l'implémentation sur les résultats
  (cas du pilote de bus SPI sur la transmission des trames)
- Premières comparaisons avec des tests sur matériel réel
  $\rightarrow$ convergence avec nos résultats de simulation / d'émulation

Limites des algorithmes actuels

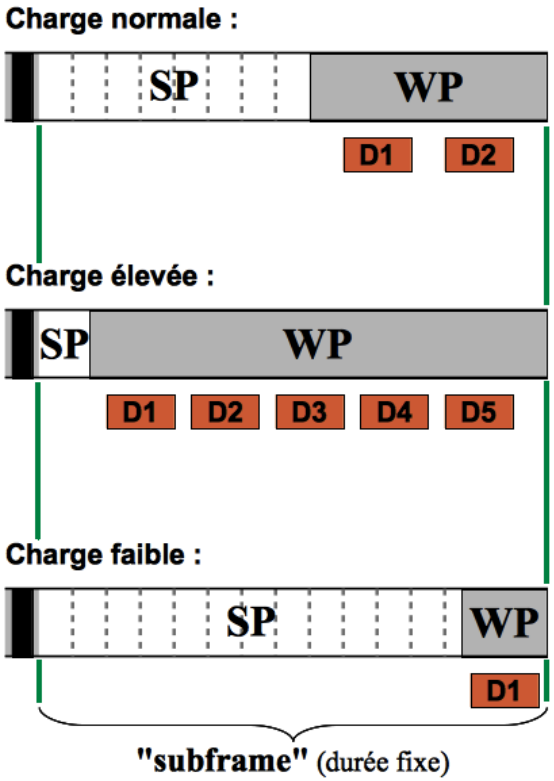
Adaptation sub-optimale aux protocoles MAC / RDC avancés

Cas de S-CoSenS :

Trafic estimé par ***comptage des paquets transmis*** :

$$\overline{\mathrm{WP}_n} = \alpha \cdot \overline{\mathrm{WP}_{n-1}} + (1-\alpha) \cdot \mathrm{WP}_{n-1}$$
$$\mathrm{WP}_n = \max(\mathrm{WP}_{min}, \min(\overline{\mathrm{WP}_n}, \mathrm{WP}_{max}))$$

- indicateur imparfait (ne voit pas tout le trafic)
- $\rightarrow$ nécessité d'augmenter WP_{min} artificiellement (cf. *duty cycle* du routeur $\geq 50\%$ au transparent 34)
- $\rightarrow$ idée d'amélioration : meilleur indicateur...



Améliorations algorithmiques des protocoles MAC I

Propositions d'améliorations algorithmiques

Proposition d'améliorations algorithmiques des protocoles MAC / RDC

1. ***Adaptation du mécanisme de "phase lock"*** de ContikiMAC aux autres protocoles aysnchrones (surtout si cycles de durée fixe)
2. ***Comptage des SFD***[1] — au lieu des paquets retransmis — pour mieux ***évaluer*** le trafic réseau (par ex. : cas de S-CoSenS)
3. Étude de ***l'influence du rapport signal / bruit (SNR)*** sur les performances des protocoles MAC / RDC de tous types

1. **SFD :** *Start-of-Frame Delimiter* (Signal court suivant le **préambule**, marquant / délimitant le début d'une trame MAC 802.15.4)

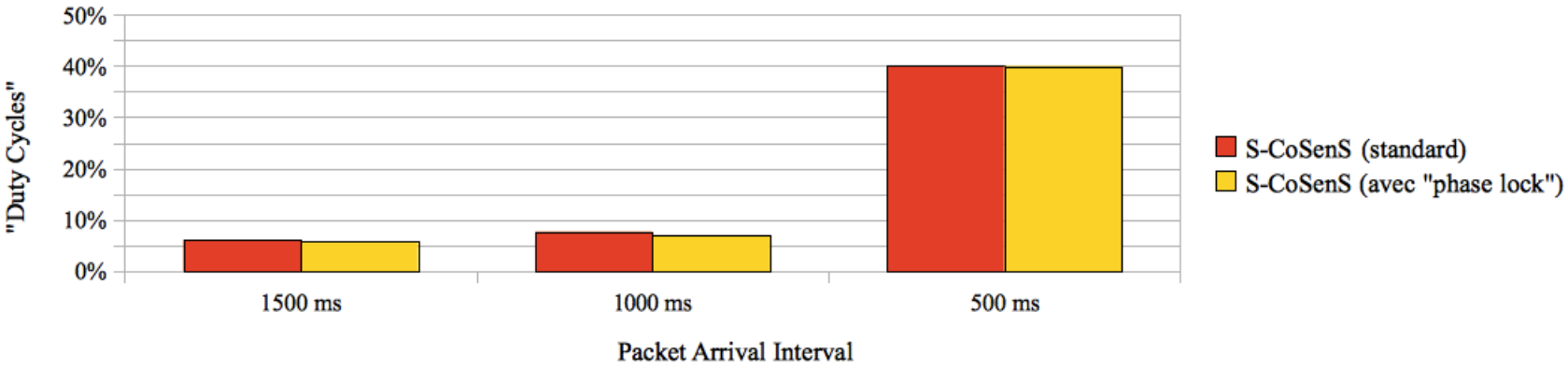
Améliorations algorithmiques des protocoles MAC II

Premières évaluations par simulation / émulation des améliorations proposées

Première évaluation des propositions d'améliorations

- Évaluations impossibles par simulation / émulation pour (2) et (3)
- Première évaluation ***exploratoire*** via simulation / émulation par Cooja pour (1) avec S-CoSenS

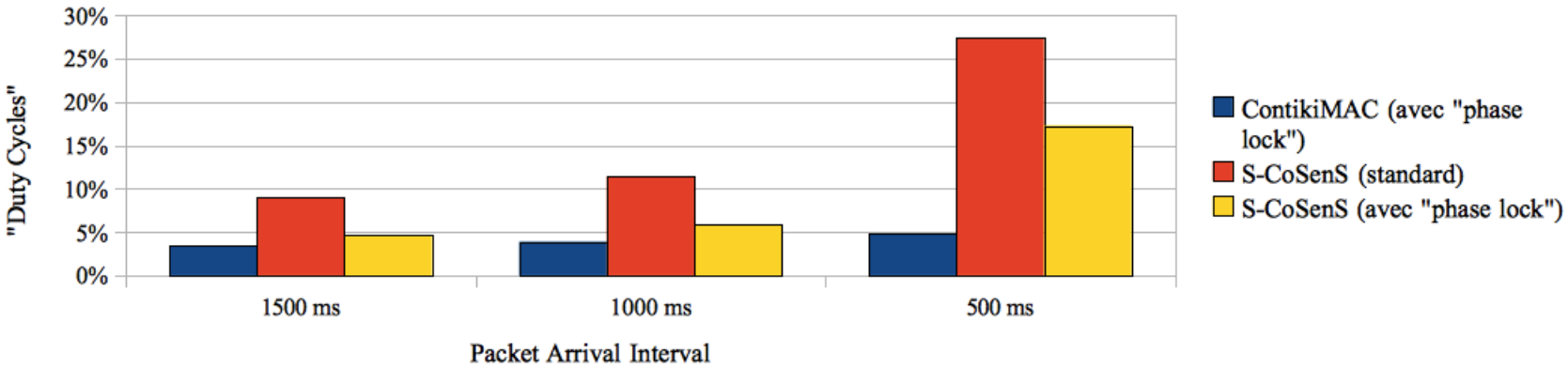
Avec une durée de cycle courte (31 ms), pas d'amélioration significative :



Améliorations algorithmiques des protocoles MAC III

Résultats des simulations / émulations pour l'adaptation du *"phase lock"* à S-CoSenS

- Avec une durée de cycle assez longue (125 ms), baisse significative du *duty cycle* de S-CoSenS, mais...
- ContikiMAC garde toujours un *duty cycle* significativement inférieur



Conclusion

Mécanisme de *"phase lock"* mieux adapté aux protocoles à durée de cycle fixe (comme ContikiMAC)

Validation des expérimentations sur plates-formes réelles — sommaire

Anomalies temporelles constatées

Problème au chargement du *buffer* d'envoi de la radio

Différences constatées entre simulations / émulations sous le *framework* Cooja / MSPSim, et tests sur matériel réel :

OS	Matériel	Différence moyenne
Contiki OS	SkyMote / TelosB	13,0 % val. exp.
	Zolertia Z1	110,3 % val. exp.
RIOT OS	SkyMote / TelosB	16,3 % val. exp.
	Zolertia Z1	183,3 % val. exp.

$\rightarrow$ simulations inadaptées aux évaluations de performances (notamment temporelles)

Conséquences potentielles

Conséquences des imprécisions / erreurs dans les simulations

Problèmes de validité des travaux basés sur simulations / émulations pour évaluer les performances :

- dans les diverses publications sur le domaine des WSN
- concernant nos propres travaux de thèse précédents

→ Nécessité de valider sur plates-formes réelles : matériels (*"motes"*) suffisamment instrumentés

Essais préliminaires sur matériel à base MSP430

→ Tendent à valider les conclusions de nos simulations précédentes

Projets de tests de validation I

Premier but : valider nos expériences précédentes

- Validation de notre implémentation de S-CoSenS
- Évaluation de ses performances via comparaison avec ContikiMAC

Projets de tests de validation II

Second but : tester la montée en charge :

→ Réseau étendu, comportant 4 PAN et 5 routeurs

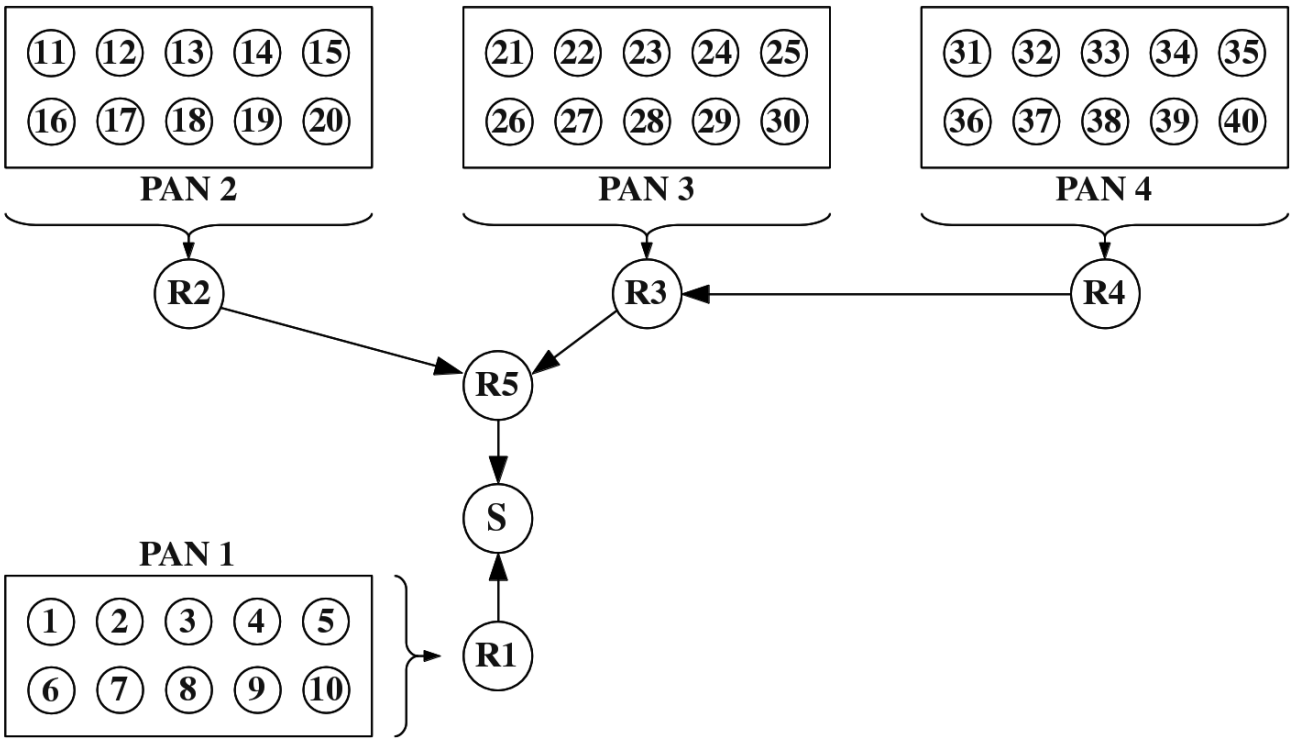


Plate-forme de tests

Choix de la plate-forme matérielle d'expérimentation

- Besoin d'un *testbed* offrant suffisamment de nœuds disponibles, ***avec possibilités d'instrumentation*** adaptées
- Choix retenu = plate-forme **IoT-LAB**, nœuds de type M3, car :
  - performants (MCU ARM Cortex-M3 → puissance de calcul et espace mémoire supérieurs)
  - assez instrumentés (*sniffer* de paquets avec horodatage, consommation énergétique, sortie console...)

Problème constaté sur IoT-LAB :

Sniffer de paquets pas assez précis pour toujours détecter les trames d'acquittement (**ACK**) !

Problèmes techniques rencontrés

Fin des travaux de validation dans le cadre de la thèse

- Multiples problèmes techniques
  → principalement « surdité » récurrente des nœuds
- Pas d'explication satisfaisante
  - bogue logiciel ?
  - problème matériel (*"brownout"*) ?

  malgré plusieurs mois d'essais et de déboguage infructueux
- Tests de validation impossibles à mener à terme
- Notre réaction et contribution : ***description technique aussi détaillée que possible des problèmes rencontrés, afin de faciliter la résolution de ces problèmes dans des travaux ultérieurs.***

Conclusions et perspectives — sommaire

Contributions I

Résumé des contributions de la thèse...

Multiples contributions sur plusieurs domaines :

- OS spécialisés dans les WSN / couches basses de leurs piles protocolaires réseau :
  - évaluation des principales plates-formes logicielles disponibles
  - contributions techniques : travail sur une plate-forme logicielle pour implanter fonctionnalités et robustesse requises
  - démonstration de la nécessité de fonctionnalités temps-réel
- Idées d'améliorations algorithmiques des protocoles MAC / RDC
- Démonstration : tout exécutable ELF (dont RIOT OS) peut fonctionner sous Cooja

Contributions II

Résumé des contributions de la thèse (suite et fin)

Multiples contributions sur plusieurs domaines (suite et fin)

- Découverte et description d'un bogue de *timing* dans MSPSim
  → code de test publié sur Github
- Limites des simulations / émulations
  → nécessité de valider les évaluations sur matériel réel
- Description aussi détaillée que possible des problèmes techniques rencontrés et non résolus dans le manuscrit
  → But : permettre la résolution des difficultés et la réussite des expériences prévues dans des travaux ultérieurs

Conclusion générale

Conclusions de notre Thèse :

Nous avons démontré :

- ***la possibilité d'implémenter des protocoles MAC / RDC :***
  - ***s'adaptant dynamiquement aux trafics réseau,***
  - ***offrant les meilleurs compromis QdS / consommation d'énergie,***
  - ***privilégiant une QdS optimale ;***

  $\rightarrow$ ***de façon à la fois efficace, portable et standardisée***
- ***la nécessité d'un OS dédié WSN :***
  - ***pour portabilité, facilité de développement et standardisation,***
  - ***offrant les fonctionnalités requises (temps-réel).***

Indispensable pour traiter des données sensibles et prioritaires : cas de l'e-santé (projet LAR, détection d'urgences...) !

Perspectives I

Terminer les travaux prévus sur matériel

- Résolution des problèmes (autre plate-forme matérielle de test)
- Adaptation de nos protocoles MAC / RDC avancés (S-CoSenS, iQueueMAC) à la pile `gnrc` de RIOT OS
- Tests sur matériel :
  - validation des travaux de la thèse faits par simulation / émulation
  - évaluation des améliorations algorithmiques proposées (SFD, SNR) sur la couche MAC
  - tests de montée en charge (réseaux complexes : multi-PAN...)

Perspectives II

Influence de la PF logicielle sur la couche MAC

Influence de RIOT OS sur les performances de protocoles MAC / RDC avancés ?

→ Collaboration en cours avec S. Zhuo, pour intégrer nos travaux dans la pile `gnrc` de RIOT OS :

- modules CSMA / CA et MAC IEEE 802.15.4 en cours de tests sur une autre PF matérielle (cartes SAMR21 Xplained Pro)
  → bientôt intégrés dans RIOT ?...
- protocoles MAC / RDC avancés : développement en cours...

Constats et idées d'amélioration

Influence de la conception des composants sur les capteurs sans-fil :

- Limitations gênantes, pour les WSN, des MCU courants
  $\rightarrow$ la première : manque de RAM
- Intégration de l'émetteur / récepteur radio
  - Avantage : facilite programmation et déboguage
  - Inconvénient : pas de choix pour l'émetteur / récepteur radio

- Encore peu de composants ***spécifiques*** dédiés aux capteurs sans-fil...
- ... Mais les industriels y viennent (ex. : Atmel AVR ATmegaRFR2, TI ChipCon SensorTags...)
- Utilisation de FPGA pour développer des SoC dédiés aux WSN ? (exemple du projet NetFPGA, lancé depuis déjà 2007)

Merci de votre attention

Questions 

Annexe : Publications et réalisations — sommaire

Publications scientifiques

Conférences / *"workshops"*

- Moutie Chehaider, Kévin Roussel, Ye-Qiong Song. « Interopérabilité des réseaux de capteurs hétérogènes dans un appartement intelligent » *9èmes journées francophones Mobilité et Ubiquité*, UbiMob 2013. Nancy, France. Juin 2013.
- Kévin Roussel. « Implementing a real-time MAC protocol under RIOT OS : running on Zolertia Z1 motes » In *Workshop Internet of Things / Equipex FIT IoT-LAB*, INRIA Grenoble Rhone-Alpes, Montbonnot, France. Novembre 2014. (présentation sans actes.)
- Kévin Roussel, Ye-Qiong Song, Olivier Zendra. « RIOT OS Paves the Way for Implementation of High-performance MAC Protocols » In *Proceedings of the 4th International Conference on Sensor Networks*, SensorNets 2015, pages 5–14. ESEO, Angers, France. Février 2015.
- Kévin Roussel, Ye-Qiong Song, Olivier Zendra. « Using Cooja for WSN Simulations : Some New Uses and Limits ». In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks* — EWSN '16 ; *workshop NextMote*, pages 319–324. *Technische Universität*, Graz, Autriche. Février 2016.

Rapports de recherche INRIA (HAL)

- Kévin Roussel, Ye-Qiong Song. « A critical analysis of Contiki's network stack for integrating new MAC protocols ». 13 pages. Décembre 2013.
  Rapport de recherche INRIA n° 8776 *(CRI-Nancy Grand Est)*.
  ID hal-01202542.
- Kévin Roussel, Ye-Qiong Song, Olivier Zendra. « Practical Lessons Learned through Implementation and Performance Evaluation of Two MAC/RDC Protocols on WSN OS » 25 pages. Mars 2015.
  Rapport de recherche INRIA n° 8777 *(CRI-Nancy Grand Est)*.
  ID hal-01202664.

Réalisations techniques I
“Pull Requests” sur GitHub

Pour Contiki OS Ces deux contributions ont été refusées (mais réutilisées en partie et indirectement pour une amélioration ultérieure de l'API radio de Contiki : PR #617 de Niklas Finne et al, intégrée au code de Contiki OS en juin 2014).

- Kévin Roussel, George Oikonomou, Mariano Alvira, David Kopf, Valentin Sawadski, Joakim Eriksson, Peter A. Bigot, Adam Dunkels. PR #192 : « Extended radio drivers API ». (2013–2014).
- Kévin Roussel, Mariano Alvira, Joakim Eriksson, George Oikonomou, Oliver Schmidt, Adam Dunkels, Nicolas Tsiftes. PR #519 : « Radio api extension ». (2014).

Pour RIOT OS Toutes les contributions citées ci-dessous dans la présente section ont été acceptées et ont été intégrées (*“merged PRs”*) à la branche principale du code de RIOT OS.

- Kévin Roussel, Ludwig Knüpfer, Oliver Hahm, Thomas Eichinger. PR #408 : « Simplify msp430 headers » (2013).
- Kévin Roussel, Oliver Hahm, Ludwig Knüpfer. PR #459 : « Msp430 lpm freq » (2013–2014).
- Kévin Roussel, Oliver Hahm, Kaspar Schleiser, Ludwig Knüpfer, Christian Mehlis, René Kijewski. PR #685 : « Panic » (2014).
  Best PR award of the month (Février 2014).
- Kévin Roussel, René Kijewski, Oliver Hahm, Ludwig Knüpfer, Christian Mehlis. PR #687 : « Add a `reboot()` function to `kernel.h` definitions » (2014).
- Kévin Roussel, René Kijewski, Kaspar Schleiser. PR #689 : « Portable definition of function attributes » (2014).
- Kévin Roussel, René Kijewski, Oliver Hahm, Ludwig Knüpfer. PR #724 : « Reboot » (2014).
- Kévin Roussel, René Kijewski, Oliver Hahm. PR #881 : « Ensure that stack pointer is correcty aligned during thread creation on MSP430 » (2014).

Réalisations techniques II

“Pull Requests” sur GitHub

- Kévin Roussel, Oliver Hahm, Thomas Eichinger. PR #882 : « CC2420 radio tranceiver's driver fixes » (2014).
- Kévin Roussel, Oliver Hahm, Ludwig Knüpfer, Christian Mehlis, Thomas Eichinger, Hauke Petersen. PR #893 : « Zolertia Z1 port for RIOT OS » (2014).
- Kévin Roussel, Ludwig Knüpfer, Thomas Eichinger, Oliver Hahm, René Kijewski. PR #915 : « Add a standard way to query CCA status on CC2420 transceiver » (2014).
- Kévin Roussel, Oliver Hahm, Ludwig Knüpfer, Thomas Eichinger, Martine Lenders, Hauke Petersen. PR #925 : « Proposal for common 802.15.4 radio driver API definition » (2014).
- Kévin Roussel, Oliver Hahm. PR #954 : « Fix for CC2420 radio driver for TelosB » (2014).
- Kévin Roussel, Oliver Hahm, Ludwig Knüpfer. PR #957 : « Handle race conditions preventing MSP430 timers to be set correctly » (2014).
- Kévin Roussel, René Kijewski, Kaspar Schleiser, Ludwig Knüpfer, Oliver Hahm, Christian Mehlis. PR #970 : « core : Add the ability to send a message to the current thread's message queue » (2014).
- Kévin Roussel, Ludwig Knüpfer, René Kijewski, Oliver Hahm, Christian Mehlis, Kaspar Schleiser. PR #1002 : « Enhance implementation of `hwtimer_ spin()` » (2014).
- Kévin Roussel, Oliver Hahm, Ludwig Knüpfer. PR #1113 : « Use Timer B on MSP430 architecture » (2014).
- Kévin Roussel, Oliver Hahm. PR #1211 : « Completing low-level radio driver definition » (2014).
- Kévin Roussel, Oliver Hahm, Thomas Eichinger, Christian Mehlis. PR #1223 : « Modify & extend CC2420 driver to comply with API described in `radio_ driver.h` » (2014).
- Kévin Roussel, Oliver Hahm. PR #1239 : « Add a missing constant in `'radio_ tx_status_t'` enum » (2014).

Réalisations techniques III

“Pull Requests” sur GitHub

- **Kévin Roussel, René Kijewski, Hauke Petersen, Ludwig Knüpfer, Christian Mehlis, Oliver Hahm. PR #1380 : « Reset ARM Cortex-M3 MCUs before flashing » (2014).**
- **Kévin Roussel, Ludwig Knüpfer, Oliver Hahm. PR #1383 : « Fix a design error in `cc2420_do_send()` function » (2014).**
- **Kévin Roussel, Ludwig Knüpfer, Oliver Hahm. PR #1385 : « Fix a nasty race condition in CCA determination on CC2420 » (2014).**
- **Kévin Roussel, Oliver Hahm. PR #1388 : « boards/z1 : fix `cc2420_txrx` function in CC2420 driver HAL » (2014).**
- **Kévin Roussel, Ludwig Knüpfer, Hauke Petersen, Kaspar Schleiser. PR #1617 : « Ensure `hwtimer_spin()` won't wait for an unreachable stop counter value » (2014).**
- **Kévin Roussel, Ludwig Knüpfer, Oliver Hahm, Hinnerk van Bruinehsen. PR #1618 : « Fix `thread_yield()` on MSP430 platforms » (2014).**
- **Kévin Roussel, Ludwig Knüpfer, Oliver Hahm, Hinnerk van Bruinehsen, Martine Lenders. PR #1619 : « Only use 16 significative bits for MSP430 `hwtimers` » (2014).**
- **Kévin Roussel, Oliver Hahm, Ludwig Knüpfer. PR #2214 : « Msp430 misc interrupt-related fixes » (2014).**
- **Kévin Roussel, Jonas Remmert, Oliver Hahm PR #4138 : « Add a `netopt` for getting and setting CCA threshold » (2015).**

“That’s all Folks!”