A critical analysis of Contiki's network stack for integrating new MAC protocols

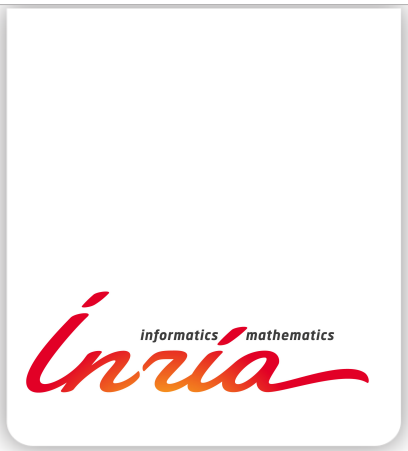
Kévin Roussel, Ye-Qiong Song

A critical analysis of Contiki's network stack for integrating new MAC protocols

Kévin Roussel, Ye-Qiong Song

RESEARCH REPORT
N° 8776
December 2013
Project-Team Madynes

ISSN 0249-6399 ISRN INRIA/RR--8776--FR+ENG



A critical analysis of Contiki's network stack for integrating new MAC protocols

Kévin Roussel, Ye-Qiong Song

Project-Team Madynes

Research Report n° 8776 — December 2013 — 13 pages

Abstract: Recent Wireless Sensor Network (WSN) MAC protocols focus on both very low idle traffic duty-cycle and high throughput when bursts of traffic occur. It is highly desirable to be able to integrate them into a common and open platform, not only for easing their performance comparison, but also for their effective interaction with existing higher-layer protocols.
As part of our work on WSN, we have implemented a new Media Access Control (MAC) protocol into the Contiki OS. When doing so, we stumbled into various limitations and quirks, relative to this system—and especially its network stack.
The present report summarizes the critical issues we have faced, and gives some ideas to fix them or work around them. Considering the widespread use of Contiki and its netstack, we believe that knowing those issues will be helpful for other researchers and developers.

Key-words: WSN, MAC protocols, Contiki OS, network stack

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Analyse critique de la pile réseau de Contiki pour l'intégration de nouveaux protocoles MAC

Résumé : Les protocoles MAC récents pour réseaux de capteurs sans-fil (WSN) sont conçus pour assurer à la fois une activité radio ("duty cycle") minimale durant les périodes d'inactivité sur le médium radio, et un débit maximal quand des pointes de trafic réseau ont lieu.

Il est hautement préférable d'intégrer ces protocoles dans une plate-forme commune et ouverte, non seulement pour faciliter la comparaison de leurs performances, mais l'efficacité de leur interaction avec les protocoles existants dans les couches supérieures de la pile réseau.

Durant nos travaux sur les WSN, nous avons implémenté un nouveau protocole MAC (*Media Access Control*) au sein de Contiki OS. Ce faisant, nous avons fait face à diverses limitations et difficultés techniques propres à ce système — et notamment à sa pile réseau.

Le présent rapport résumé les problèmes critiques auxquels nous avons fait face, et donne quelques idées pour les régler ou les contourner. Eu égard à l'usage généralisé qui est fait de Contiki et de sa pile réseau, nous pensons que de telles informations seront utiles aux autres chercheurs et développeurs.

Mots-clés : réseaux de capteurs sans-fil, protocoles MAC, Contiki OS, pile réseau

1 Introduction

Wireless Sensor Networks (WSNs) currently form a very active topic of research. One aspect of this topic is the fast-paced development of WSN-oriented operating systems. Two projects clearly stand out in that domain: **TinyOS** [1], and **Contiki** [2]. Both are open-source projects, providing free[1] systems to the WSN—and more generally embedded devices—oriented community of scientists and developers. We chose the latter as our software platform, since it is written in standard C language (whereas TinyOS uses a specific derivative named "nesC"), and seems to be—as of now—more actively used and supported by both scientific and industrial communities.

Another important aspect of WSN research is the optimization of **Quality-of-Service (QoS)** and **energy consumption** directly into the network stack mechanisms, especially at the **Media Access Control (MAC)** level. This is an aspect on which many teams are actively working: the standard **IEEE 802.15.4 MAC protocol** [3], with its static, fixed duty-cycle, has been challenged by many other MAC protocols. These alternative MAC protocols try to improve upon the standard by providing dynamic runtime adaptation to the network traffic load, better QoS, and minimized energy consumption (by allowing radio transceivers to be turned off as much as possible); most of these alternatives are adaptive duty-cycled protocols.

Adaptive duty-cycled protocols can be classified into many families: some of the best-known protocols, like S-MAC [4] and T-MAC [5], are member of the synchronous MAC protocols family. Others, like B-MAC [6], X-MAC [7], RI-MAC [8], and WiseMAC [9] protocols, belong to the asynchronous MAC protocols family. (Other families can be named, like frame-slotted, or multichannel MAC protocols.) Among the asynchronous protocols, we can distinguish between those based on the Low-Power Listening (LPL) technique—such as B-MAC and X-MAC—and those using Low-Power Probing (LPP)—like RI-MAC. The creator of the Contiki OS has himself designed and published a MAC protocol, named ContikiMAC [10], that is now the default protocol used by the Contiki OS. This protocol is largely inspired from X-MAC and WiseMAC, and as such is also an asynchronous, LPL-based, adaptive duty-cycled protocol. Finally, ContikiMAC has itself served as a basis for further improvements, notably the recent and promising Strawman protocol [11].

[1]Here, the term "free" is used as a synonym of liberty (of use and diffusion), as in the "free speech" expression.

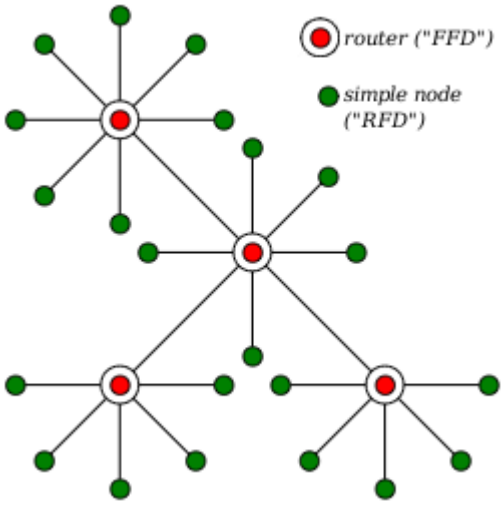


Figure 1: An example of network using 2-tier, star-like topology.

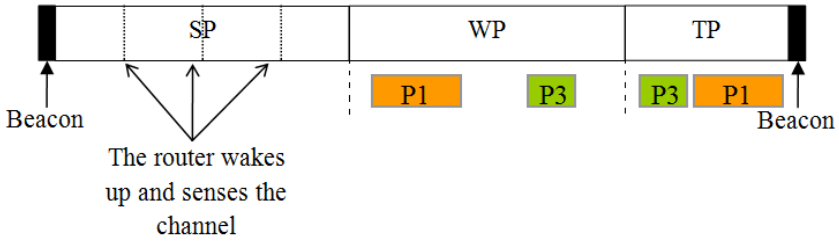


Figure 2: The basic working principle of the CoSenS protocol.

Our team, Madynes[2] at INRIA, is also active in this domain: we have already presented many high performance traffic-adaptive MAC protocols for WSNs; among these stand the **CoSens** [12] protocol which has been implemented on Jennic motes, and iQueue-MAC [13] which has been implemented on STM32W108 chips.

Although an actual implementation becomes nowadays almost a must step to validate any new WSN MAC protocols, their specific implementation does not allow a fair comparison. Worser, it is very hard to use them within a full stack of protocols if routing or upper-layers are required. So as already explained before, integrating those new protocols into a well-accepted platform, such as Contiki, becomes highly desirable.

In this paper, we provide a critical analysis of important issues on implementing new MAC protocols on Contiki network stack. This analysis shows

[2]MADYNES team: MAnaging DYnamic NEtworks and Services, `http://madynes.loria.fr/`

the limitations of actual version of Contiki and proposes improvements.

The simplicity of implementation has driven us to use CoSenS as a first element of choice to implement into the Contiki network stack. More evolved protocols, like iQueue-MAC, are planned to be implemented in Contiki as soon as possible (as a second step).

Before detailing the issues we faced during this implementation work, we will briefly summarize the workings of the CoSenS protocol.

2 The CoSenS protocol: Quick Summary

Since the CoSenS protocol is already explained and detailed in [12], we will here only quickly summarize its design and features, and thus explain why we choose it as a first stepping stone in our exploration of Contiki's internals.

CoSenS is designed for networks built upon a classical 2-tier, star-like architecture (an example of which is represented on figure 1) that can be, where one can distinguish between **router nodes**—that can be assimilated to the "Full-Function Devices" (FFD) cited by the 802.15.4 standard [3]—and **simple, "leaf" nodes**—assimilable to 802.15.4 "Reduced-Function Devices" (RFD). Simple nodes' role is mainly to perform measurements thanks to their built-in sensors, and are supposed to possess only limited networking abilities: they can only communicate with the router node they are associated with—a simple node is always associated with one and only one router that acts as its "parent" in the network. Conversely, the routers run full-fledged network stacks, and are consequently responsible for packet-routing duties through the network; thus, each router act as a gateway for a variable number of simple nodes, that are designated as the router's "children".

CoSenS is designed to run on the router nodes, while the simple nodes are supposed to use the classical CSMA/CA protocol. They are always in sleeping mode unless they have packets to send. CoSenS is derived from that classical CSMA/CA protocol, improving significantly on it in terms of performance for medium and high traffic loads, and thus facilitating the implementation of QoS mechanisms.

The basic idea of CoSenS is the following: the duty cycle of a router node is divided into three periods: a **Sleep Period (SP)** where the router goes to sleep but periodically wakes up and senses the channel to receive its neighboring router's packets, a reception period where the router passively waits for packets to be received—this period is thus named **Waiting Period (WP)**—, and a **Transmission Period (TP)**, during which it will transmit the packets it received. Indeed, when a router receives a packet (during a WP), it won't retransmit it right away; that packet will be en-

queued, for deferred retransmission during the next TP. During these TPs, enqueued packets are sent in "burst-mode", avoiding the overhead related to the channel acquisition procedure of the CSMA/CA protocol for all but the first sent packet (which will use CSMA/CA, followed by a preamble similar to that of B-MAC). Routers' duty cycles, under CoSenS, simply consist in the succession of SPs (sleep for saving energy), WPs (for receiving packets) and TPs (for retransmitting them).

That simple mechanism (a "Sleep, Collect then Send burst scheme", as its author calls it) that can be summarized by figure 2, gives nevertheless a significant performance advantage to CoSenS over both the standard 802.15.4 MAC and the classical CSMA/CA protocols. Moreover, its simplicity of design and implementation has naturally driven us to choose it as the first protocol we would try to implement, as a beginning of our work into Contiki's network stack.

3 Working with the Contiki Network Stack

While implementing the CoSenS protocol as an addition to the Contiki's network stack (**"netstack"**), we discovered many issues that hinder development into Contiki, both in general and at the specific level of the netstack. We will now detail these issues, and—when possible—make propositions to fix or at least work around these problems.

3.1 Lack of documentation

This is a recurrent problem with Contiki: apart from source code comments and examples, there is actually almost no documentation. This concerns all of Contiki in general, and as such its netstack in particular.

One can especially point the total absence of technical reference documents, where the general architecture of the system, as well as design and implementation choices, could be explained; such reference documents quickly become key tools to really understand a software platform, and thus to be able to become a proficient developer.

There is also a serious shortage on tutorials: the only official introductory document is the "get started" web page, that will quickly show you how to use the provided "Instant Contiki" Linux distribution to start simulations (using the project's simulator named **Cooja** [14]) and download program examples to hardware. No document will teach you and (more importantly) explain you the various features of the Cooja simulator, no document will show you how to program applications on Contiki, what are the specificities

of embedded development (when compared to "classical" programming for PCs)—even less how to touch its internals! Such absence makes the initial approach to Contiki for new developers difficult, by further heightening an already high learning barrier.

The main source of documentation, when it comes to Contiki development, is the "contiki-developers" mailing list[3]: any person wishing to do development on this platform (be it at the applicative or at the system level) just won't be able to achieve any real goal without subscribing to it, and requesting informations and help from its members. While this list provides a rich source of knowledge, is reactive, and friendly welcomes newcomers, it can hardly be considered as a satisfactory replacement for the missing technical and introductory documentation mentioned above.

Let's also add that, if there are many examples provided of application-level code using Contiki OS, there is no such examples of code working at the system-level (i.e.: designed to plug into Contiki's internals). This kind of development is thus even more difficult to apprehend and learn.

3.2 Limitations

Many limitations found in Contiki were dicted by the constraints imposed by the devices that constitute WSNs: they are indeed very limited when it comes to processing power and (especially) memory space.

Nevertheless, one of them is just a "missed point" that can be completed quite easily with some additional code.

We will begin the following list with this missing element, then examine deeper problems, that are the direct consequences of some basic design choices made for the Contiki netstack.

3.2.1 Missing features: incomplete radio drivers

As of the current version (release 2.7), the API for the radio transceivers drivers only provides the most basic features: sending and receiving a packet, turning transceiver on and off, and checking for channel availability (CCA). Access to all other features of transceivers (e.g.: frequency/channel switching, transmission power setting, changing node address(es), etc.) forces to directly access the transceiver, thus breaking portability and separation between netstack layers.

[3]Contiki-developers mailing list:
address: `contiki-developers@lists.sourceforge.net`
management URL: `https://lists.sourceforge.net/lists/listinfo/contiki-developers`

To solve this problem, we have submitted an API extension, allowing to access these extended features via a standard and flexible "capabilities" mechanism, without breaking compatibility with existing code, and incurring only minimal overhead (addition of three mere pointers to a radio driver definition). The three added function pointers serve respectively to 1) query the radio transceiver for built-in configuration constants (e.g.: maximal power level for packet emission), 2) to get and 3) to set configuration parameters—a.k.a. "capabilities"—impacting the behaviour of the radio transceiver (e.g.: radio channel, node address(es), etc.). We are currently waiting for the response of the project's maintainers to our contribution.

3.2.2 Netstack centered on an unique "packetbuf"

One very specific feature of Contiki's netstack is that it is centered around a unique packet buffer (named **"packetbuf"**), that constitutes the mandatory "work area" for the various netstack layers (see below).

This design pattern was chosen to spare precious memory: on wireless sensor devices, program code must indeed fit in dozens of kilobytes of flash RAM, while data—and among them, network packets—is restricted to very limitative amounts of RAM (sometimes nothing more than 2 kilobytes!).

Unfortunately, this design choice has some undesirable consequences:

- packet loss: since that unique buffer is to be used by *all and every part of the netstack, at every time*, any breach in the synchronization between the various netstack components may result in overwriting the "packetbuf" at the wrong time, thus provoking data loss! The risk of a packet incoming (i.e. being received by the radio transceiver) during the processing of another data by the netstack, is especially critical, since the arrival of external data is, by nature, an event that code (even at system-level) cannot plan nor avoid.

- unabilty to handle queues efficiently: designing efficient MAC protocols almost always implies to handle packet queues (both in emission and reception); while Contiki also provides mechanisms for them ("queuebuf", "packetqueue"), the unique "packetbuf"-centered design means unceasing copies between that "packetbuf" and the queues, that is: waste of time, processing power, and even memory—being able to designate a portion of memory (i.e.: a given packet in a queue) as the current work area at runtime would actually use less memory than the current "unique packetbuf" scheme. Also note that the provided "queuebuf" and "packetqueue" mechanisms have themselves a complex and

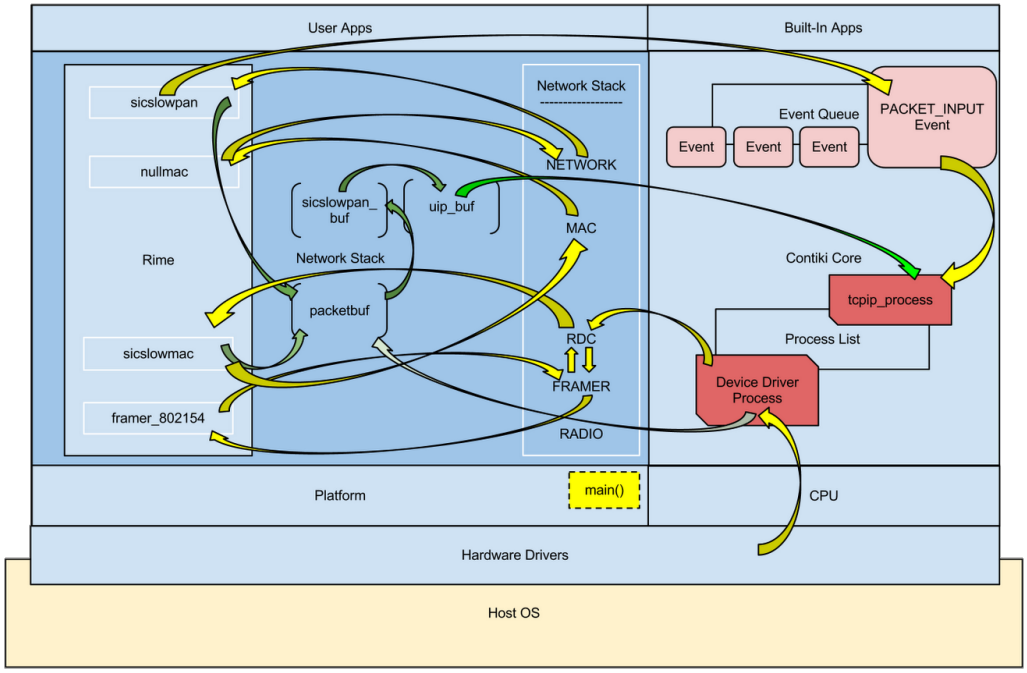


Figure 3: Functional representation of Contiki netstack's layered architecture (from [15]).

counter-intuitive design and use, because they need to work with the unique "packetbuf".

- netstack complexity: the need to handle correctly that unique packet buffer actually contributes to make the netstack code more complex, since the various elements of the netstack need to perform various operations that can become quite complex—like careful synchronization, or never-ceasing copies to/from packet queues—that wouldn't be needed on a more flexible (e.g.: queue-based) design.

3.2.3 Separation of MAC and RDC layers

The Contiki netstack is designed around the principle of layer separation. There are indeed distinct drivers for radio transceivers, routing protocols, etc.

As a part of this strategy, there is a distinction, in Contiki, between the **Radio Duty Cycle (RDC)** layer—that is supposed to control how the radio transceiver is turned on and off during duty cycles, thus implementing the energy-saving strategy—and the **Media Access Control (MAC)**

layer—that is (theoretically) only tasked with ordering and sequencing packet transmissions.

In practice, most modern MAC protocols do manage both of these aspects, in order to adapt dynamically network efficiency (QoS) *and* energy consumption to the ongoing traffic (e.g.: [7] [8] [12] [13], among others). This separation consequently becomes artificial, and only adds more unneeded complexity to MAC protocol implementations, since separating these two aspects is at best difficult, and often impossible: as an example, the ContikiMAC protocol is itself implemented as a sole RDC driver (preferably used with a "null" MAC driver).

3.2.4 Excessive complexity of the netstack

As a generalization of the previous point, most of the Contiki netstack suffers from excessive "layerization": there is indeed no less than five levels involved (from bottom to top):

1. RADIO: the "physical" driver controlling the radio transceiver;
2. FRAMER: the parser/generator of formatted packets (e.g.: "`framer_802154`" for the 802.15.4 protocol);
3. RDC: the Radio Duty-Cycle controller (see above);
4. MAC: the Media Acess Control protocol implementation;
5. NETWORK: the network-level protocol implementation (e.g. 6LoWPAN).

If such a design can (in theory) provide more flexibility in the implementation, it actually add its complexity to the other constraints of the netstack: namely, the unique "packetbuf" architecture, the difficulty to clearly separate between layers (see the previous point), the need to interact smoothly with other parts of the system—especially the event and the protothread [16] subsystems, that allow Contiki to offer lightweight cooperative multitasking features.

The layered architecture of Contiki network stack can be summarized by figure 3 (from [15]); one can easily see on this figure the complexity induced by the netstack's design choices.

Combined with the lack of available documentation (especially technical references), this all makes contributing code to that netstack a difficult, tedious, and error-prone task.

Moreover, since these difficulties are direct consequences of netstack's fundamental design principles, fixing them is not envisionable without reworking most of the system's internals, which would imply major (and most likely unacceptable) compatibility breaks for the system-level code. The need for a precise, helpful, detailed technical documentation is thus even more critical.

4 Conclusion

We have summarized, in the present paper, the issues we had to face when trying to implement a new MAC protocol into Contiki's network stack; these issues—especially the lack of technical documentation—are most likely to be encountered by every developer undertaking the development of system-level code into Contiki OS (and probably even, at a lesser degree, development of application-level code).

As a attempt to bring solutions to these problems, we have taken different actions: we have proposed code contributions designed to add missing features (especially at the radio drivers level); and we have identified some problematic points that developers are bound to meet, thus paving the way towards a comprehensive set of technical documentation and tutorials we wish to contribute, at a later point, to the Contiki project.

Thus, we hope, by facilitating the task of developers, to contribute to widen even further the diffusion and use of the Contiki, as we believe that this OS has not only already proven to be a very valuable asset to the WSN research and development community, but also has still much to offer.

Acknowledgment

The authors would like to thank the *Living Assistant Robot* and the *ANR Quasimodo* (ANR grant # 2010 INTB 0206 01) projects, for providing fundings to the research work corresponding to the present paper.

References

[1] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An Operating System for Sensor Networks," in *Ambient Intelligence*. Springer Berlin Heidelberg, 2005, pp. 115–148, `http://www.tinyos.net/`.

[2] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *IEEE 29th Conference on*

Local Computer Networks, ser. LCN '04. IEEE Computer Society, 2004, pp. 455–462, `http://www.contiki-os.org/`.

[3] IEEE 802.15.4, "IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pp. 1–314, 2011.

[4] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," in *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communication Societies*, ser. INFOCOM 2002, vol. 3. IEEE, 2002, pp. 1567–1576.

[5] T. V. Dam and K. Langendoen, "An Adaptive Energy-efficient MAC Protocol for Wireless Sensor Networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys '03. ACM, 2003, pp. 171–180.

[6] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04. ACM, 2004, pp. 95–107.

[7] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '06. ACM, 2006, pp. 307–320.

[8] Y. Sun, O. Gurewitz, and D. B. Johnson, "RI-MAC: A Receiver-initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in Wireless Sensor Networks," in *Proceedings of the 6th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '08. ACM, 2008, pp. 1–14.

[9] A. El-Hoiydi and J.-D. Decotignie, "WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks," in *Proceedings of the Ninth International Symposium on Computers and Communications*, ser. ISCC 2004, vol. 1. IEEE, 2004, pp. 244–251.

[10] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," Swedish Institute of Computer Science, Tech. Rep. T2011:13, 2011.

[11] F. Österlind, L. Mottola, T. Voigt, N. Tsiftes, and A. Dunkels, "Strawman: Resolving Collisions in Bursty Low-power Wireless Networks," in *Proceedings of the 11th International Conference on Information Processing in Sensor Networks*, ser. IPSN '12. ACM, 2012, pp. 161–172.

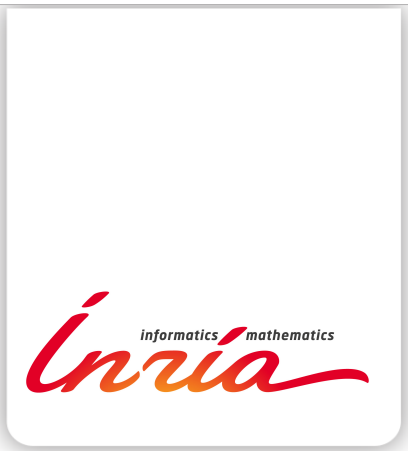
[12] B. Nefzi and Y.-Q. Song, "QoS for Wireless Sensor Networks: Enabling Service Differentiation at the MAC Sub-layer Using CoSenS," *Ad Hoc Networks*, vol. 10, no. 4, pp. 680–695, June 2012.

[13] S. Zhuo, Z. Wang, Y.-Q. Song, Z. Wang, and L. Almeida, "iQueue-MAC: A Traffic Adaptive duty-cycled MAC Protocol with Dynamic Slot Allocation," in *IEEE 10th Conference on Sensor, Mesh, and Ad Hoc Communications and Networks*, ser. SECON 2013. IEEE Communications Society, 2013, pp. 95–103.

[14] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with Cooja," in *IEEE 31st Conference on Local Computer Networks*, ser. LCN '06. IEEE Computer Society, 2006, pp. 641–648.

[15] J. Udit, "Jeremy's blog – Contiki OS article," [Online: personal blog of Jeremy Udit], August 2012, available at: `http://jeremyudit.blogspot.fr/2012/08/contiki-os.html`.

[16] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '06. ACM, 2006, pp. 29–42.



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique

CS20101

54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399