

Portability versus Efficiency Tradeoffs in MAC Implementations for Microsensor Platforms

Anthony Schoofs, Peter van der Stok, Phillip Stanley-Marbell

Abstract—Medium Access Control (MAC) implementations control access of network devices to a transmission medium. For emerging communication protocols, the MAC is typically implemented in software, to enable adaptation to evolving de-facto or industry standards. Software MAC implementations are typically realized as state machines, executing code related to successive MAC states within periodic interrupts. This software construct yields minimal memory footprint and energy efficiency, but the resulting implementations are often tightly coupled to the platform’s system software, and are thus non-portable across hardware and system platforms. This article presents an architecture that decouples MAC and system software, enabling portability, while preserving software efficiency.

Index Terms—Embedded Systems, Personal Area Networks, Medium Access Control, Sensor Networks

I. INTRODUCTION

MICROSENSOR platforms have scant resources, making the development of embedded software a cumbersome task. Many of the challenges these systems pose are however mutually contradictory: **Portability and modularity**—the requirements of hardware platforms in emerging embedded systems domains evolve rapidly, prior to the achievement of industry standards. This requires software *modularity* and *portability* to facilitate quick adaptation; **Efficiency**—software *efficiency* in terms of memory footprint, network and computation latency, and energy, is required as systems are almost always under severe constraints for these resources; **Flexibility**—low-level system software (e.g., device drivers) need to be ported to the hardware platforms of interest. Integration of components is very dependent on the software’s *flexibility* (versus fragility); small changes in the implementation of one component should not lead to needs for significant reorganization of other components.

Medium Access Control (MAC) implementations control access of network devices to a transmission medium, and are one of the major system components in wireless microsensor platforms. This article presents an implementation architecture that decouples MAC and system software, enabling portability, and flexibility, while preserving software efficiency.

II. BACKGROUND

For emerging communication protocols, the MAC is typically implemented in software, to enable adaptation to evol-

ing de-facto or industry standards. Software MAC implementations are often realized as state machines, executing code related to successive MAC states within periodic network event or timer interrupts. This software organization yields minimal memory footprint and good energy efficiency, but the resulting implementations are often tightly coupled to the platform’s system software. They are thus not easily portable across processor architectures and system platforms. MAC software requires accurate timing behavior to function correctly. When MAC software is tightly coupled with system software, and, for example, embedded within interrupt handlers, the timing behavior of other time-critical system components may be adversely affected.

An alternative design methodology, is to implement the MAC state machine as an application task, whose execution is not embedded within timer and network peripheral interrupts. Leaving an interrupt-driven software architecture creates several opportunities for improving portability, modularity, and flexibility. Furthermore, as previously observed in a related context [1], this can be achieved without sacrificing efficiency.

Appropriate software organizations may actually improve energy efficiency. For example, in wireless low-power microsensor platforms, a *radio transceiver* integrated circuit implements the interfacing between software and the wireless channel, and is typically the largest source of power dissipation. The energy consumed by the transceiver is related to the size and generation frequency of data packets/frames. Data processing on the node (e.g., signal processing, data aggregation, etc.) can be used to intelligently reduce the number of packets to be transmitted and thus potentially lower the energy consumption of the whole system [2] while improving timing performance.

A. Processing of interrupts in low-power control processors

The event-driven nature of computing systems is particularly pronounced in microsensor platforms, where the primary activity of systems is to react to the occurrence of temporal or sensing events. In virtually all systems, the change of normal program control flow in response to such events is through an *interrupt mechanism*, in which program execution (or idling) is halted, and the processor jumps to a pre-determined address to execute a new stream of instructions. The ability to return to its previous execution flow is aided by saving execution state, and possibly, prevention of the occurrence of further interrupts. If a processor can accept interrupts during interrupt processing, it is said to be capable of taking *nested interrupts*. It is possible for microcontrollers to receive more than one interrupt during the same cycle, generated asynchronously from

A. Schoofs is with CLARITY: Centre for Sensor Web Technologies, University College Dublin, Dublin, Ireland. e-mail: anthony.schoofs@ucdconnect.ie.

P. van der Stok is with Philips Research, Eindhoven, The Netherlands. email: peter.van.der.stok@philips.com.

P. Stanley-Marbell is with IBM Research GmbH, Zurich Research Lab, Switzerland. email: pst@zurich.ibm.com.

peripheral devices or synchronously from on-board timers. Thus, in addition to interrupt nesting, interrupt prioritization is important. Higher priority interrupts cannot be preempted by lower priority ones, reducing the risk of unwanted delays.

B. IEEE 802.15.4 MAC Basics

The IEEE 802.15.4 standard [4] defines physical and MAC layers for low bit-rate and low-power consumption wireless personal area networks (LR-WPANS). The transmission of MAC frames relies on a *carrier sense multiple access with collision avoidance (CSMA/CA)* protocol to schedule communication, which may be either *slotted* or *unslotted*. The slotted CSMA/CA employs a *superframe* structure, which defines a division of time into slots (spanning the duration of multiple frames). Nodes compete for channel access (and thus, for slot access) during *contention access periods (CAPs)*. There are also *guaranteed time slots* reserved for special purposes; *beacons* are used in the slotted version of the CSMA/CA algorithm to define the CAPs. In the absence of beacons, the MAC sub-layer uses the unslotted version of the CSMA/CA algorithm. In both cases, the algorithm is implemented using units of time called backoff periods, where one backoff period is equal to $320\ \mu\text{s}$ [4].

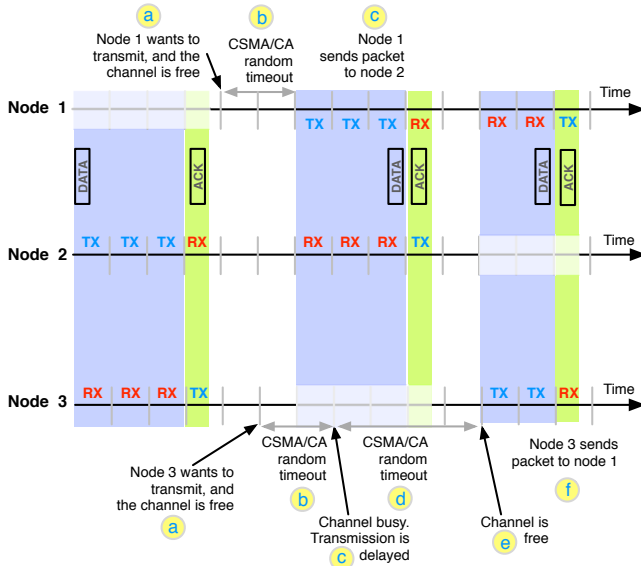


Fig. 1. Transmission of frames using carrier sense multiple access with collision avoidance (CSMA/CA).

The MAC sub-layer ensures that the *physical layer (PHY)*, implemented in the radio transceiver hardware, starts transmission on the boundary of a backoff period, as shown in Figure 1. This requires accurate timing (e.g., a periodic event to be generated on the microcontroller), and is one factor affecting MAC software implementation portability and efficiency.

III. MICROKERNELS FOR MICROSENSOR MACS

To address the challenges discussed in Section I, an IEEE 802.15.4 MAC implementation was developed, based on the idea of a preemptive task execution environment. The design's objectives were to:

- 1) decouple application software from MAC execution;
- 2) avoid dependency on specific hardware features;
- 3) achieve software efficiency while facilitating reusability.

To achieve these objectives, a system architecture was implemented in which MAC tasks are activated by a microkernel scheduler.

A. The dual scheduling approach

When an application requires several processes to be executed with different timing characteristics, dividing the application into functions which are executed by independent tasks simplifies implementation. The proposed MAC implementation provides the opportunity to have a microkernel task execute the MAC protocol with a task status controlled by MAC events and timer interrupts (henceforth, *ticks*). The MAC task, like the other tasks in the node, is scheduled on the basis of the timer interrupts of the operating system (OS) timer. Additionally, timers are used in the MAC for time-stamping network and sensing events, and for channel access at backoff slot boundaries. Thus, when the MAC is executed within a microkernel environment, the OS needs to provide OS ticks for generic tasks, facilities for event capture, and MAC ticks to schedule MAC functions.

These timing facilities are supported in most traditional microcontrollers by *timer engines*, hardware units with features such as compare and timer capture registers. Not all microcontrollers embed such a timer engine, however. For example, the CoolFlux DSP [5] is equipped with two timers that only provide snapshot and single compare register features. To cope with different timing and hardware support across existing platforms, a *dual scheduling* approach was developed. One timer generates OS ticks for the microkernel scheduling while a second one generates ticks for the MAC scheduling when needed; on hardware platforms with sophisticated timer engines, a single hardware timer engine can generate the two ticks.

The key idea behind the dual-scheduling approach is the separation of timing control for the MAC from the timing control for other tasks in the microkernel. The microkernel configures a tick interrupt at a chosen timer interrupt frequency, independently for the MAC and for other tasks. The tick frequency for applications can thus be dynamically adapted to avoid unnecessary overhead and reduce power consumption. Similarly, the MAC timer can be deactivated whenever communications are not needed.

B. Implementation

The dual-scheduling microkernel approach for efficient and portable coexistence of MACs and applications was implemented using the IEEE 802.15.4 MAC, and the FreeRTOS microkernel [6], on the SAND hardware platform [7]. The SAND node combines the CoolFlux DSP and the TI CC2420 radio transceiver [8] in a microcontroller + transceiver configuration.

In the microkernel-based implementation of the MAC, the MAC is considered as an application whose duty is to drive the transceiver to be compliant with the IEEE 802.15.4 standard.

Accordingly, the MAC will be run by one FreeRTOS task, which we call the *FreeRTOS MAC task*. This task is given a high priority to ensure meeting timing constraints of the IEEE 802.15.4 standard.

Applications that need to access the MAC use a set of primitives defined by the 802.15.4 standard. For that purpose MAC primitives called by higher layers will add MAC requests to one of four priority queues, as shown in Figure 2. The FreeRTOS MAC task executes the corresponding functions by taking the oldest request from the highest priority non-empty queue, and executes the corresponding function, starting on a backoff slot boundary.

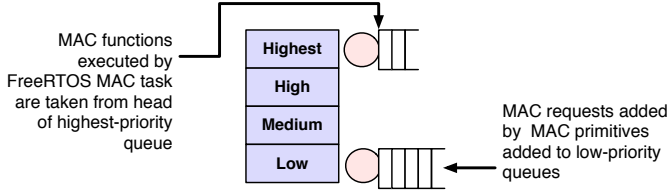


Fig. 2. Execution of MAC functions as requested by MAC primitives.

As long as the priority queues are filled, the MAC timer is active and generates interrupts every $320\ \mu\text{s}$. Upon interrupts from the CC2420 transceiver, the FreeRTOS MAC task executes the MAC functions according to the contents of a *task information structure*, which contains information on the currently active functions and the associated packets. Figure 3 summarizes the successive steps executed for processing packet transmission and reception. The different software triggers are highlighted to emphasize the software constructs and the relationship between the components of the software system.

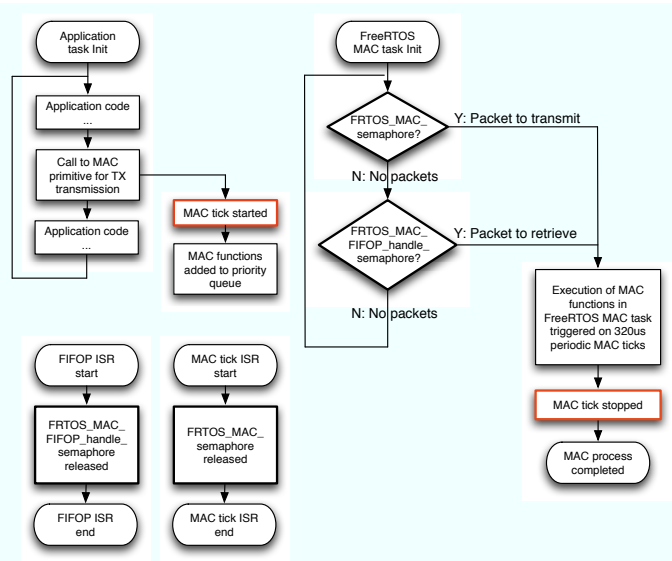


Fig. 3. Triggers and relationships between software components for processing packet transmission and reception.

IV. PERFORMANCE AND PORTABILITY EVALUATION

A *hardware abstraction layer* was used to decouple the MAC's hardware needs from the facilities provided on a

specific platform; additional details are available in a previous report [9]. The original implementation was performed for the CoolFlux DSP, and this was ported in three months to a TelosB-like node (having a TI MSP430 microcontroller and a TI CC2420 transceiver). The majority of the effort in the MSP430 port involved adapting to a new compiler and dealing with the new architecture's timer peripheral. Due to the lower clock frequency of the MSP430, a large part of the period was used to optimize the medium access mechanisms under heavy transmission load, which led to significantly different runtime behavior compared to the CoolFlux DSP.

A. Integration and Reuse

The proposed dual scheduling design is currently used for a wide range of applications. The benefits of the design allowed for fast prototyping of new applications with different requirements such as timing and process control, via the reuse and integration of software components. For example, a stroke rehabilitation application [10] required synchronized data sampling on different nodes to meaningfully correlate data at the monitoring station. Time synchronization and distributed task synchronization as described in [12], [13] have also been successfully implemented within the presented software environment. An emotion sensing application [11] used these facilities gratefully. The dual-scheduling approach permitted easy integration of application software with no conflicts from the MAC process execution on the task execution. Finally, a lightweight IP stack with a 6LoWPAN adaptation layer have been implemented on top of the software system [14].

B. Memory usage

The current version of the MAC software includes support for both beacon and non-beacon networks. Two 128 B buffers for outgoing and incoming packets are used. The code size is approximately 11 KB and the data usage is approximately 2.3 KB. In comparison, the source code base from which the implementation was derived had a data memory footprint of 21 KB for code and 2 KB for data, with an additional 3 buffers of 200 B each (albeit on a different processor architecture).

Part of the reason for the code and data memory usage reduction is the Target Compiler [15], which was especially designed for the CoolFlux architecture.

C. Latency and throughput

The best useful throughput (MAC payload throughput) achieved on the SAND platform is 86 Kb/s. Each packet transmission included an 11 B MAC header, a 2 B MAC footer, and a 60 B MAC payload. Including the MAC header in the calculation, the throughput reached 105 Kb/s. This result is far below the theoretical upper bound of 220 Kb/s as given in [16]. Our results are lower than the throughput of 140 Kb/s achieved by [16] with a Tmote Sky node running the Contiki Operating System in similar conditions. As another comparison, a data rate of 127 Kb/s with Jennic JN5121 wireless modules was reported elsewhere [17]. To understand the origin of the lower throughput, we analyzed our software constructs to determine

the mechanism responsible for the poor throughput, or whether the results are intrinsic to the proposed dual scheduling design or to its implementation.

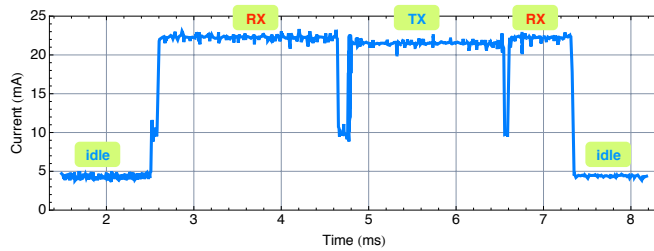


Fig. 4. Measured energy and time duration involved in the different steps of a packet transmission realized with the presented task-based MAC design.

The authors of [16] claim that the bottleneck in single-hop transfer on microsensor platforms is neither the operating system nor clear-channel assessment, but rather, *serial peripheral interface (SPI)* packet copying. SPI data transactions are required between the microprocessor and the transceiver to load the TX-FIFO with data, and can also be triggered in the receive path by interrupts, or by sampling the status of the interface (i.e., polling). Figure 4 shows a timing measurement realized on the SAND platform, with an interrupt-triggered SPI transaction. The duration of the SPI transaction is approximately 2 ms and represents one fourth of the total transmission duration. In an interrupt-triggered SPI transaction process, overhead comes from the context switch happening on each SPI interrupt and from the access to memory to fetch the next byte to transmit. Often, the overhead introduced by this method is longer than the transfer time. Consequently, for throughput purposes it is significantly faster to use polling.

The interrupt-based implementation starts RF transmission when all data has been transferred from the microprocessor to the transceiver. With a polling approach for SPI transmission, we are able to start RF transactions when the first bytes of data are received at the transceiver, thus reducing the latency and improving the overall throughput.

The advantage of using an interrupt driven approach is the decoupling of the application code from the communication code, because calls to hardware are done asynchronously. The consequence is an interleaved execution of transmission and application thus increasing the processor utilization as less processor cycles are wasted on active waiting cycles during polling. Because of the above-mentioned consequences we made both interrupt-triggered and polling SPI control available to the application developer, for an optimization on either latency/energy/throughput or process multitasking according to application requirements. In our implementation we had chosen to optimize processor efficiency at the expense of communication latency and throughput. If available on a given microcontroller, DMA facilities could be used to further reduce CPU overhead, thereby increasing the SPI throughput.

The FreeRTOS context switch latency on the CoolFlux DSP was measured to be 17 ms. During transmission using SPI polling, FreeRTOS context switches happen at each MAC tick interrupt, every 320 μ s, and at each FreeRTOS tick interrupt, every 1 ms. With approximately four ticks per millisecond, a

FreeRTOS context switch of 17 μ s represents an overhead of about 7% on the available processing power, whenever the network stack is active; when the MAC queues are empty, and no MAC ticks are required, this is reduced to less than 2%. Overall, the SPI throughput performance depends on the processor architecture and SPI interface mechanism.

V. CONCLUSION

This article presented the design and implementation of a MAC architecture for LR-WPANs, in which MAC tasks are activated by a microkernel scheduler. This approach is in contrast to the typical interrupt-triggered MAC implementations, and cleanly decouples the MAC implementation from that of the system software. Such a design enables portability and flexibility, while preserving software efficiency.

ACKNOWLEDGMENT

The authors thank Albert Rietema and Peter van der Meer for their work on the software implementation of the MAC, and Elmahi Adam for his efforts on porting the MAC to the MSP430. Sheau Pei Chong performed the latency and throughput measurements. This work is partially financed by the European Commission under the Framework 6 IST Project Wirelessly Accessible Sensor Populations (WASP) and supported by Science Foundation Ireland under grant 07/CE/I1147.

REFERENCES

- [1] R. Min and A. Chandrakasan, *Mobicom poster: top five myths about the energy consumption of wireless communication*, *SIGMOBILE Mob. Comput. Commun. Rev.* 7, 1, 65–67, 2003.
- [2] B. Krishnamachari, D. Estrin, S. B. Wicker, *The Impact of Data Aggregation in Wireless Sensor Networks*, Proceedings of the 22nd International Conference on Distributed Computing Systems, 2002.
- [3] Jones, Nigel. *Minimize your ISR overhead*, Embedded Systems Programming, January 2007.
- [4] IEEE 802.15.4 Standard-2003, *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for LR-WPANs*, 2003.
- [5] H. Roeven, J. Coninx, M. Ade, *CoolFlux DSP - The embedded ultra low power C-programmable DSP core*, GSPx 2004.
- [6] R. Barry, FreeRTOS™, <http://www.freertos.org/>
- [7] M. Ouwerkerk, W.F. Pasveer, N. Engin, *SAND: a modular application development platform for miniature wireless sensors*, BSN 2006, USA.
- [8] Texas Instruments, CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver.
- [9] M. Andree (ed.) et al., *Core Hardware Abstraction and Programming Model*, WASP European IST-034963 project public report, 2008.
- [10] R. D. Willmann et al., *Home Stroke Rehabilitation for the Upper Limbs*, Proceedings of the 29th Annual International Conference of the IEEE EMBS Cite Internationale, Lyon, France, August 23-26, 2007.
- [11] J.H.D.M. Westerink et al., *Probing Experience - From Assessment of User Emotions and Behaviour to Development of Products*, Philips Research Book Series, Vol. 8, 2008.
- [12] M. Aoun, A. Schoofs, P. van der Stok, *Efficient Time Synchronization for Wireless Sensor Networks in an Industrial Setting*, ACM Sensys 2008, November, 2008.
- [13] M. Aoun et al., *Distributed Task Synchronization in Wireless Sensor Networks*, EWSN'09, February, 2009.
- [14] A. Schoofs et al., *IP-based Testbed for Herd Monitoring*, IPSN'09, 2009.
- [15] CoolFlux DSP C Programmer's Manual PLR-15542 version 5.0, Target compiler distribution, September 2005.
- [16] F. Osterlind and A. Dunkels, *Approaching the Maximum 802.15.4 Multi-hop Throughput*, Proceedings of HotEmNets 2008, 2008.
- [17] Jennic, *Calculating 802.15.4 Data Rates*, Application Note: JN-AN-1035, 2006.