Datalogisk Institut
KØBENHAVNS UNIVERSITET

Never Mind the Standard Here is the TinyOS 802.15.4 Stack

Jan Flora & Philippe Bonnet

Technical Report no. 06/10
ISSN: 0107-8283

Dept. of Computer Science
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

Never Mind the Standard:

Here is the TinyOS 802.15.4 Stack

Jan Flora
University of Copenhagen
janflora@diku.dk

Philippe Bonnet
University of Copenhagen
bonnet@diku.dk

Abstract

802.15.4 has been designed to support wireless monitoring and control applications with tight energy budget and limited throughput needs. This sounds promising, but can this standard be efficiently implemented on 8-bit sensor motes with limited resources and computing capabilities? Ideally, it should be possible to trim off parts of the stack that are not relevant for a given sensor profile in order to reduce memory footprint. More generally, it should be possible to adapt the stack to the hardware limitations of a given platform. In order to study these issues, we cannot use the implementations provided by radio manufacturers. These are fairly undocumented blackboxes without tuning capabilities. We are thus developing an 802.15.4 stack in TinyOS on a Motorola HCS08 microcontroller and Freescale MC13192 radio. In the process, we are experiencing the wide gap that exists between the specification of a standard and a real implementation. In this paper, we report on the two most challenging issues we faced: timing and cluster tree network support. We argue that the timing requirements of the IEEE 802.15.4 standard is not supported properly by the proposed OSI reference model design and we propose a new PHY/MAC interface. We also question the ability of the standard to support cluster tree networks topology.

1. Introduction

Two of the barriers to the adoption of wireless sensor networks are their price and their complexity. The emergence of standards could help lower these barriers. In particular, the IEEE 802.15.4 protocol has been designed for applications with tight energy budgets and limited throughput needs and should thus match the characteristics of a large class of wireless sensor networks. The IEEE 802.15.4 standard has been around for 3 years this summer, and there are only few actual deployments.

One of the problems is that existing 802.15.4 stacks are blackboxes provided by radio constructors. It is thus impossible to reduce the memory footprint of the 802.15.4 stack, or to tune it to the meet the constraints of a given hardware platform (e.g, the 802.15.4 based Chicpon 2430 SoC that relies on a 8051 processor).

In order to tackle these issues, we decided to implement an 802.15.4 stack in TinyOS.

TinyOS is an open-source embedded operating system for networked sensor platorms. Using an event based execution model, every task is scheduled using a simple task queue. Only code initiated by hardware interrutps may preempt running tasks. TinyOS trades less than 1KB of memory for a highly modular system with a variety of hardware platforms. The strength of TinyOS lies in its component-based approach, introducing easy-to-comprehend wiring of components into applications. This makes code recycling easy and allows for a natural code division using interface definitions. One of the questions we want to study is whether the modularity of TinyOS can help trim the memory footprint of the 802.15.4 stack.

We are developing a generic IEEE 802.15.4 protocol stack for TinyOS on a Motorola HCS08 microcontroller and Freescale MC13192 radio. Our implementation is

available on SourceForge[1]. We leave the evaluation of our implementation (in term of memory footprint, throughput and stability), the lessons learnt implementing timing constraints in TinyOS, as well as a discussion of the tuning issues for future work[2]. In this paper we focus on the main issues we faced in the design and implementation phases: timing and cluster tree networks support.

Note that an implementation of 802.15.4 in TinyOS has been claimed to be impossible [5]. This is true for the modular design dictated by the standard. However, it turns out that none of the radio manufacturers actually follow the standard. We document the inconsistencies of IEEE 802.15.4 in terms of timing and cluster tree network establishment. We propose to redefine the PHY/MAC boundary in 802.15.4 to actually accommodate the design of existing 802.15.4 radio chips. We explore the design space for cluster tree networks.

2. Timing is Everything

When reading through the IEEE 802.15.4 standard, it is clear that timing is of the essence. Nearly all operations at the PHY or MAC layer have to conform to timing requirements. However, overall timing considerations are nowhere to be found. Take for example the calculation of CRC for incoming frames. According to the IEEE 802.15.4 standard the CRC calculations are done in the MAC layer, meaning that a frame has to travel through the PHY layer before the frame can be considered valid. There are several problems with this approach:

> Time is wasted passing the frame through the PHY and finally calculating the CRC in the MAC. A frame that passes CRC check also has to be passed through additional filters to ensure that it is destined for the receiving device [1].
>
> A frame cannot be processed until proven valid. Time constraints for time critical operations might be broken.
>
> CRC calculations could be performed incrementally at a lower layer. The CRC can actually be calculated in hardware during frame reception so that invalid frames can be discarded right away.

IEEE 802.15.4 radio chip manufacturers like Freescale and Chipcon have acknowledged these problems and are providing CRC calculations as part of their hardware design [4][6]. Chipcon even has address decoding and destination filtering in hardware [4]. The constructors of 802.15.4 radios thus break the standard.

2.1 Timing of Acknowledgements

The most time critical operation in the 802.15.4 stack is the acknowledgement of incoming frames. According to the standard these acknowledgements should be implemented at the MAC layer [1].

An acknowledgement frame is to be transmitted within 192 μs[3] after receiving the last byte of a frame [1]. Thus, the time used by the radio hardware to switch from receive mode to transmit mode must be at most 192 μs (known as the turnaround time [1]). As a result, the worst case scenario is that the transmission of an acknowledgement frame happens immediately after receiving the last frame byte. Both the Freescale MC13192 and the Chipcon CC2420 have a turnaround time that is below the maximum turnaround time allowed: 144 μs for the MC13192 and 128 μs for the CC2420. This leaves us with respectively 48 or 64 μs to pass the frame to the MAC layer, perform CRC checking and destination filtering, generate the acknowledgement frame, pass the acknowledgement frame back to the PHY layer and start the transmission. With a typical clock rate of 16 MHz and a CPI of around 4, we have to be able to accomplish the above task in

[1] http://tinyos.cvs.sourceforge.net/tinyos/tinyos-1.x/contrib/diku/evb13192/

[2] Please check Jan's blog at http://nflora.dk/studie for regular updates on the evaluation. You can also consult the archive to read about the daily challenges faced during the design and implementation phases.

[3] The timing calculations are based on a symbol period of 16 μs corresponding to the symbol period used for a 2.4 GHz radio.

192 to 256 instructions. This must be considered an impossible task. Alone destination filtering requires more instructions.
From the above calculations it should be clear that both CRC checking and destination filtering need to be done while receiving the frame. Transmission of acknowledgement frames must be initiated immediately after receiving the frame needing acknowledgement. We suggest that all of these operations are moved to the PHY layer to be as close to the hardware as possible.

2.2 Radio Operation Timing

The nature of the IEEE 802.15.4 synchronization, collision avoidance and TDMA schemes require some radio operations like frame transmissions to be timed precisely. Poor timing will result in poor performance due to loss of frames and possibly loss of synchronization. Even though the timing is so crucial, the MAC/PHY interface suggested in the standard does not mention any timing information at all. This means that the timing lies implicitly in the execution time of the function call. The timing of the radio operation is thereby determined by (i) the execution time of the PHY function and (ii) the hardware initiation delay. This is not adequate if we want a generic implementation of the MAC layer to work with PHY layers from several different radio chips.
Additional timing information is also needed when transmitting frames using slotted CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance). Both the CCA (Clear Channel Assessment) operations and the transmit operation need to be aligned to a so called backoff boundary for the CSMA-CA scheme to be effective. The backoff boundaries are 320 μs apart. Flooding the PHY layer with CCA and transmit requests periodically with a period of 320 μs will possibly require operation queuing capabilities in the PHY layer. We suggest making the request for CCA an attribute of the transmission request. This will make obsolete the explicit CCA request in the PHY interface and move more critical timing issues to the PHY layer. This also matches the decision from the hardware designers, as both the MC13192[4] and the CC2420 radio implement transmissions with CCA in hardware.

2.3 A Revised PHY/MAC Interface

We can now present a simplified and timing aware PHY/MAC interface for 802.15.4. The new interface leaves all critical timing issues to be handled differently on each platform by the platform specific PHY layer. As an example, the MC13192 chip leaves many of the timing critical operations to be handled in software, while the CC2420 chip handles almost everything in hardware including auto acknowledgement. The new PHY/MAC interface only has 4 operations:

Transmit frame/Transmit done
Additional attributes:
Commence time
CCA requested
Enable receiver/Data ready
Additional attributes:
Commence time
Energy detect/Detection done
Disable transciever

The first three operations can be thought of as request/confirm pairs. The attributes listed are additional to those attributes specified for the corresponding operations in the IEEE 802.15.4 standard.
Apart from the radio operations we operate with the following state variables:

Current channel (set)
Contention window size (set)
Transmit power (set)
Ack backoff alignment (set)
Supported channels (get)

All variables that are written (set) from the MAC layer do not need to be read (get), since the value is already part of the MAC layer state. The channels supported by the radio chip is never written from the MAC as this corresponds to a hardware abstraction.

[4] This is actually an undocumented feature controlled in some of the hidden registers of the MC13192 chip. The feature is hinted in the Freescale 802.15.4 PHY source, which is available from the Freescale website.

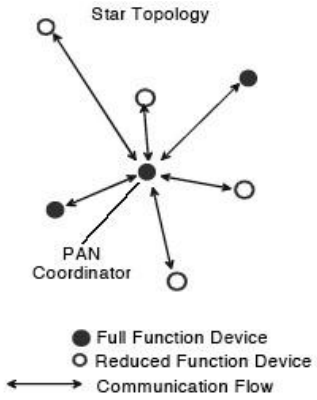

Figure 1: Star Topology

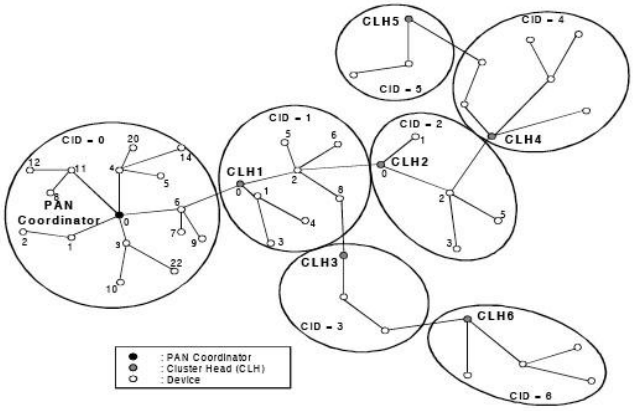

Figure 2: Cluster Tree Topology

3. Cluster tree networks

The IEEE 802.15.4 standard aims at supporting larger scale networks of up to 65536 nodes. In contrast to the simple star topology network (see Figure 1), such a network will require more that just a single PAN coordinator. In a star topology network, only two device roles are necessary: (1) The PAN coordinator - typically represented by a FFD (Full Function Device) and (2) the end devices - typically represented by one or more RFD (Reduced Function Devices).

The synchronization is dictated by the PAN coordinator as it sends out network beacons[5] for the end devices to adjust their timing contexts. The network timing context is called a superframe. The network beacon indicates the beginning of the superframe and the beacon interval determines the time between beacons. The superframe is used to define the timing of the next beacon transmission, the CAP (Contention Access Period) in which a CSMA scheme is used, the CFP (Contention Free Period) in which a TDMA (Time Division Multiple Access) scheme is used and the inactive period (see Figure 3). The end devices are then able to communicate with the coordinator in the specified periods.

When extending a network from a star network to a cluster tree network (Figure 2), a new device role is needed: The coordinator. A coordinator replicates the attributes of its parent coordinator, thus acting as a substitute for the network PAN coordinator when forming a separate network cluster. In a cluster tree network, such a PAN coordinator replica is often referred to as a cluster head. There are several issues involved with creating a coordinator, none of which are addressed in the IEEE 802.15.4 standard. The most serious issues are: (1) superframe clashing and (2) operating in two superframe contexts

Some of these problems are mentionned in the Freescale 802.15.4 User Guide [2] and in the Zigbee 1.0 specification [3].

3.1 Avoiding Superframe Clashing

The first step when becoming a coordinator node is associating to an existing coordinator or PAN coordinator (referred to as the parent coordinator). After a successful association the node acts as a normal end device. To become a coordinator, the node must start transmitting its own network beacon, thus starting a new cluster. The question of, when the new coordinator should transmit its beacon is not addressed by the IEEE 802.15.4 standard. We cannot accept overlaps between child and parent coordinator superframes since the two coordinators are within communication range. A superframe overlap would interfere with the TDMA scheme and would create unnecessary channel contention. To solve this problem, the parent coordinator

[5] We only consider beacon enabled networks in this section, as cluster tree networks cannot be formed without using network beacons.

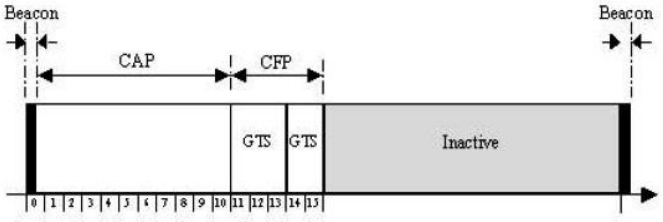

Figure 3: The Superframe Structure

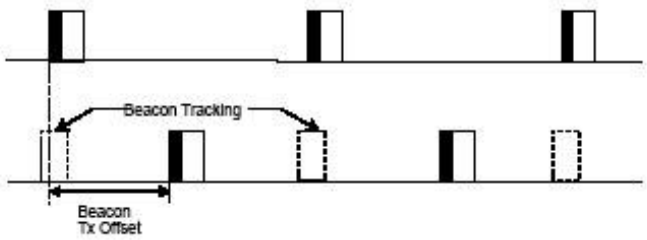

Figure 4: Superframe Alignment

superframe must be shorter that the parent coordinator beacon interval. The new child superframe can then be placed in between consecutive parent superframes, utilizing the inactive period as shown in Figure 4. To counteract potential drift issues, the child beacon time should be specified as an offset to the parent beacon time. How to achieve a good scheduling of superframes between neighboring coordinators is out of the MAC layer scope.

We needed a way for the application to pass the parent beacon offset to the MAC layer when starting up a coordinator. The Zigbee specification suggests that the start request primitive is extended with a StartTime parameter, while the Freescale approach is to add an extra state variable in the MAC layer (accessible through the PAN information base (PIB)). We chose to use the Freescale solution, as it does not change the APP/MAC interface.

3.2 Operating in Two Superframe Contexts

When taking on the coordinator role, a device has to be able to operate in two different superframe contexts: The superframe context of the parent coordinator and the superframe context of the local cluster. This raises issues regarding, which superframe context is addressed by which operation or request. The IEEE 802.15.4 standard does not touch these issues at all. As the superframe contexts only impact radio communication, we need to determine in which superframe context a particular frame is to be transmitted, and in which superframe context the receiver needs to be turned on, following a particular request. Luckily all frames but data frames are implicitly tied to a specific superframe context, as they are only sent from coordinator to end device or vice versa. Accordingly problems only arise in the following situations:

When turning on the receiver. Either (i) úsing the RxEnable request primitive, or (ii) due to RxOnWhenIdle PIB attribute being set
When transmitting data

Let us look at the receiver problems first. The RxEnable primitive is very useful when doing peer to peer communication and will typically be used to force devices not acting as coordinators into receive mode for a certain period of time. As the RxOnWhenIdle PIB attribute is typically used to keep the receiver enabled on coordinators during the CAP, the RxEnable request primitive is less valuable to the coordinator. We chose to let the RxOnWhenIdle PIB attribute refer to the local superframe con-

text, and let the RxEnable request primitive operate in the parent superframe context. That way, an application is able to enable the receiver in both superframe contexts in a well defined way.

The data transmission problem does not have such an elegant solution. The problem is caused by a lack of information with regards to cluster members. We need to know whether the addressee of a given data frame resides inside or outside of the local cluster. If we do not have any addressing information for members of our cluster, it is impossible to know the superframe context in which a data frame should be transmitted. We can only determine that frames destined for the parent coordinator must be transmitted in the parent coordinator superframe context. One possible solution is to let the coordinator register the addresses of all associated devices, in order for the coordinator to keep track of its own cluster. Since cluster member information is already registered in the higher layers, this solution introduces information redundancy. Also since all decisions regarding capacity of a cluster is decided above MAC level, buffer space for storing member addresses in the MAC layer could prove a potential problem.

Without information about cluster members, the best solution we can hope for is to limit the data transmission capabilities between a coordinator and its parent network. We can choose to only allow data transmission from a given coordinator to its parent coordinator in the parent coordinator superframe context. Data destined for all other addresses will be transmitted in the local superframe context. This solution imposes some restrictions on the routing possibilities between two clusters, since only routing between cluster heads would be allowed, and is thus far from optimal. Especially because cluster heads are typically chosen to reside far from each other in order for cluster ranges to overlap as little as possible. A third possibility is to alter the data request primitive to contain information about the addressee residing inside or outside the local cluster.

We went for the simple solution as we didn't want to change the APP/MAC interface for compatibility reasons, even though that clearly is the best solution. The member list solution seemed too clumsy and too big an effort to fix a problem caused by a bad interface design. This problem will hopefully be addressed in the next IEEE 802.15.4 standard revision.

4. Conclusion

In this paper we have documented some of the problems encountered when implementing an IEEE 802.15.4 protocol stack on a real hardware platform (HCS08 microcontroller and Freescale MC13192 radio). Timing issues proved to be one of the biggest hurdles and led us to suggest major changes in the PHY/MAC interface. This interface actually reflects the design choices of radio manufacturers. We are in the process of porting our 802.15.4 stack to the CC2420 radio to verify that our design is generic.

Support for the cluster tree topology is very sketchy in the current IEEE 802.15.4 standard. It seems that, just like Bluetooth, the first generation of IEEE 802.15.4 protocol stacks lacks the complete design to support multihop networks. An open source implementation of 802.15.4 will certainly help explore the design space.

5. References

[1] IEEE Std 802.15.4-2003, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)

[2] 802.15.4 MAC PHY Software User's Guide, Freescale Semiconductors, September 2005, Rev. 1.2

[3] ZigBee Specification v1.0, ZigBee Alliance, December 2004

[4] CC2420 datasheet, Texas Instruments, October 2005, Rev. 1.3

[5] Jonas Thomsen, Dirk Husemann: Evaluating the use of motes and TinyOS for a mobile sensor platform. In *proceedings of the 24th IASTED Parallel and Distributes Computing and Networks Conference*, February 2006

[6] MC13192/MC13193 Reference Manual, Freescale Semiconductors, April 2005, Rev. 1.3