TKN Telecommunication Networks Group

Technical University Berlin

Telecommunication Networks Group

TKN15.4: An IEEE 802.15.4 MAC Implementation for TinyOS 2

Jan-Hinrich Hauer

hauer@tkn.tu-berlin.de

Berlin, March 2009

TKN Technical Report TKN-08-003

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

Abstract

We present *TKN15.4*, a platform independent IEEE 802.15.4-2006 MAC implementation for the 2.1 release of the TinyOS operating system. After establishing a set of design goals we introduce the *TKN15.4* architecture, describe the component breakdown and illustrate how *TKN15.4* builds upon an extension of the TinyOS 2 resource arbitration mechanism and a set of revised PHY/MAC interfaces. While the TinyOS 2.1 implementation was developed on the Tmote Sky platform a core design goal was platform-independence: as long as a set of well-defined requirements are met the MAC implementation can be used on any TinyOS 2 platform. We give an overview of the TinyOS-2.1 implementation and explain the necessary steps for making *TKN15.4* available on a new TinyOS 2 platform.

Contents

Chapter 1

Introduction

In 2003 the task group 4 of the IEEE 802.15 working group released the first edition of the 802.15.4 standard [14] to enable wireless connectivity between "ultra-low complexity, ultra-low cost, ultra-low power consumption, and low data rate" devices in wireless personal area networks (WPAN). The standard covers the physical layer (PHY) and the medium access control sublayer (MAC) in the ISO-OSI layered network model. It has attracted strong interest in industry and academia and is now (partially) adopted by several other wireless standards, such as ZigBee [30], IETF 6lowPAN [13] or WirelessHART [9].

In contrast to the many analytical and simulation studies of the 802.15.4 MAC [22, 29, 21], as well as the several experimental investigations of the 802.15.4 PHY [23, 18, 28] so far very little research related to the 802.15.4 MAC (and protocols above) has been evaluated experimentally, with real hardware under realistic conditions. While the first steps in protocol design can often be made with the help of analytical models and simulation, the last steps require the use of real hardware, in realistic environmental conditions and experimental setups. The lack of an empirical investigation of the 802.15.4 MAC has not least been caused by the fact that a stable, open-source 802.15.4 MAC implementation has been unavailable. Our work aims at closing this gap: we present *TKN15.4*, a platform independent IEEE 802.15.4-2006 MAC implementation for the 2.1 release of the TinyOS [16] execution environment.

The 802.15.4 MAC is responsible for a number of tasks ranging from PAN association and disassociation, over periodic beacon transmission and synchronization to the actual channel access mechanism. The standard supports several different configurations: a PAN may operate in a star- or peer-to-peer topology; it may use periodic beacon transmission (beacon-enabled PAN) or not (nonbeacon-enabled PAN); and the channel access can either be contention-based (unslotted CSMA-CA), contention-free (guaranteed time slots, GTS) or scheduled contention-based (slotted CSMA-CA). The numerous configuration options in conjunction with the tight resource constraints of a typical sensor node platform call for a flexible design that can be adapted to the particular application requirements. Because it allows to break a design into many self-contained components and provides an event-based programming model that matches the 802.15.4 specification quite well, we have selected the TinyOS 2 operating system [10] as the execution environment for our 802.15.4 MAC implementation. While all development was made on the Tmote Sky platform [20], one primary design goal is platform-independence: as long as a small set of well-defined requirements are met (essentially a certain radio chip abstraction and timers that satisfy the accuracy and pre-

cision requirements of the standard) the MAC implementation can be used on any TinyOS 2 platform.

The rest of this document is structured as follows: Chapter 2 presents an overview of the IEEE 802.15.4 standard and the TinyOS 2 operating system, Chapter 3 describes the *TKN15.4* design goals and discusses practical challenges and Chapter 4 introduces the *TKN15.4* architecture including (1) a component breakdown of the MAC, (2) an extension to the TinyOS 2 resource arbitration mechanism, and (3) a set of revised PHY/MAC interfaces. The following Chapter 5 reports on the details of the TinyOS-2.x implementation, Chapter 6 presents related work and Chapter 7 concludes this document.

Please note that this document does not include an evaluation of *TKN15.4*, because we defer the evaluation, including interoperability tests with certified MAC implementations, to a future report. Note also that this report is based on the *TKN15.4* TinyOS 2 CVS[1] version of 3rd April 2009; if future modifications affect the content of this document, then this will be recorded in the *TKN15.4* `README.txt` file.[2].

[1] `http://sourceforge.net/cvs/?group_id=28656`

[2] `tinyos-2.x/tos/lib/mac/tkn154/README.txt` in the `tinyos-2.x` module

Chapter 2

Background

This Chapter gives a brief overview of the IEEE 802.15.4 standard [15] and the TinyOS 2 operating system [16].

2.1 IEEE 802.15.4

Following the general IEEE 802 policy the standard covers the physical layer (PHY) and the medium access control (MAC) sublayer.

2.1.1 Physical Layer

The PHY services are structured into a data and management plane. The PHY data service allows to transport MPDUs between peer MAC sublayer entities; the PHY management service allows to control the internal operating state of the transceiver (receive/transmit/off), to perform clear channel assessment (CCA), energy detection measurements within the current channel and to set PHY-specific attributes like the RF channel or transmit power. The standard specifies four PHYs in the 868/915/2450 MHz ISM bands based on different modulation techniques. In the popular 2.4 GHz band the direct sequence spread spectrum (DSSS) PHY achieves a data rate of 250 kb/s using offset quadrature phase-shift keying (O-QPSK). One 4-bit symbol is mapped to a 32-chip sequence and chips are transmitted at a nominal chip rate of 2 Mchip/s resulting in a symbol rate of 62.5 ksymbol/s. The transmit power of a 802.15.4 transceiver is typically around 0 dBm.

2.1.2 Medium Access Control Sublayer

The MAC services are structured into a data and management plane. The data service allows to transfer a MSDU to a peer device, which may include an acknowledgement from the peer device and/or several retransmissions. The management service is responsible for device configuration, periodic transmission of and synchronizing to beacons, enabling PAN association and disassociation, employing security mechanisms and handling the GTS mechanism.

The MAC sublayer supports different configurations and operating modes. One commonality is that every network has exactly one PAN coordinator, which is the primary controller responsible for PAN identifier and device address assignment and device synchronization.

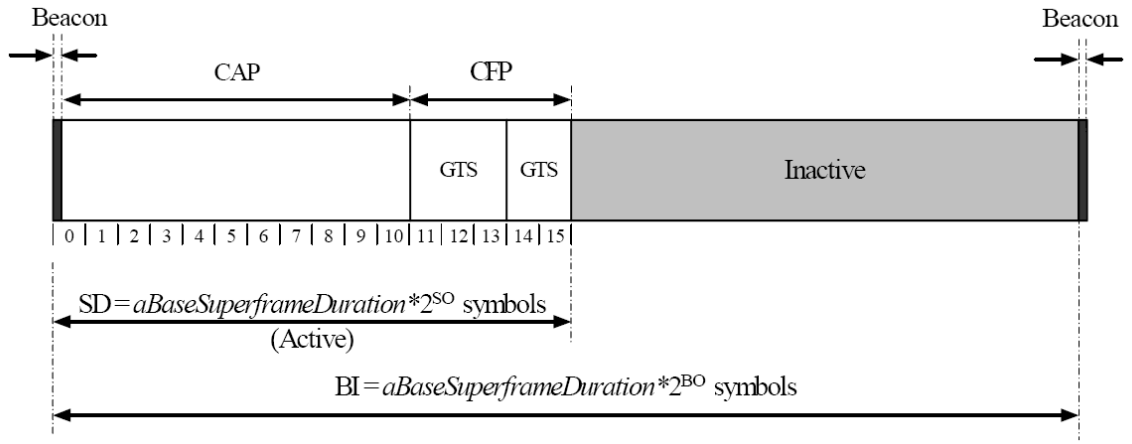


Figure 2.1: IEEE 802.15.4 superframe structure (source: [15]).

802.15.4 PANs can either be nonbeacon-enabled or beacon-enabled. In a nonbeacon-enabled PAN frames are transmitted according to an unslotted CSMA-CA algorithm (nonpersistent CSMA): if the channel is detected idle the transmission can start immediately otherwise the device defers the transmission for a random time period uniformly drawn from an exponentially increasing backoff interval. The destination device(s) either have to always listen or achieve synchronization with schemes beyond the scope of the standard.

In beacon-enabled mode coordinators periodically transmit beacons which mark the beginning of a superframe as depicted in Figure 2.1. A beacon carries information about pending data and the current network configuration. Immediately after the beacon follows the contention access period (CAP). During the CAP devices use a slotted variant of the CSMA-CA algorithm: a device must sense an idle channel twice before it may transmit and both, channel sensing and transmission must be performed on backoff slot boundaries. The CAP is followed by an optional contention-free period (CFP), which is portioned in so-called guaranteed time slots (GTS). GTSs are allocated dynamically and the corresponding time interval can be used exclusively to transmit packets in a contention-free fashion. The CFP is followed by an optional inactive period in which all nodes can sleep to preserve energy and achieve low duty cycles.

2.2 TinyOS 2

TinyOS [10] is an operating system for wireless embedded sensor networks. It consists of a set of components and the nesC language [7] allows to combine these components to create an application. TinyOS 2 [16] is the second generation of the operating system that keeps many of the basic ideas of its popular predecessor while pushing the design in several key areas.

TinyOS 2 improves system reliability and robustness by redefining some of the basic TinyOS 1.x abstractions and policies such as initialization, the task queue, resource arbitration and power management. For example, in TinyOS 2, every task has its own reserved slot

in the task queue and can be posted *only once*. These semantics lead to greatly simplified code (no need for task reposting on error) and more robust components. The same principle of compile-time allocation and binding is applied to many other aspects of the system: components allocate all of the state they might possibly need; and the invariants are explicitly reflected by the components and their interfaces, rather then being checked at runtime. This design principle limits the flexibility, but makes many OS behaviors deterministic.

TinyOS 2 supports code reusability and portability in several ways. It is based on a nesC version that uses the concept of *generic components*, which, similar to object-oriented programming, can be used to create multiple component instances, but instantiated at compile-time. It also supports *network types* at the language level: programs can declare data structures and primitive types that follow a cross-platform (1-byte aligned, big- or little-endian) layout and encoding. This allows services to specify platform independent packet formats without resorting to macros or explicit serialization. Finally, it uses a three-layer *Hardware Abstraction Architecture* (HAA) [27] that decomposes the functionality of an individual subsystem, for example timers, into three distinct layers. The HAA achieves code portability through platform independent interfaces at the top layer, but at the same time allows the applications to access a subsystem's full capabilities through the chip-specific second layer, if performance requirements outweigh the need for portability.

Chapter 3

Design Goals and Challenges

This chapter lists the *TKN15.4* design goals and discusses practical the challenges for an implementation.

3.1 Design Goals

The design of *TKN15.4* is based on three main goals: platform independence, modularity and extensibility. They are explained in the following.

3.1.1 Platform Independence

A platform independent 802.15.4 MAC implementation is decoupled from a particular mote platform and can be used on any platform that provides a compatible execution environment. There are essentially three things that a platform independent 802.15.4 MAC implementation requires from the execution environment: timers that satisfy the precision and accuracy requirements specified in the standard (e.g. 62.5 kHz and ± 40 ppm for the 2.4 GHz PHY), suitable computational abstractions (processes/tasks) and a suitable radio chip (PHY) abstraction. The first two requirements are typically satisfied with the help of the operating system, assuming that a suitable hardware clock is available. Ideally, the 802.15.4 PHY interfaces would realize the interfaces towards the physical layer (radio). Flora et al. [4], however, have argued that the specified PHY interfaces turn out to be problematic in practice: since the PHY provides only very basic services (like activation and deactivation of the radio transceiver), many time critical operations would have to be performed inside the MAC. In addition to periodic beacon transmission and tracking the most time critical operations are as follows:

- In the CAP portion of a superframe the slotted CSMA-CA algorithm requires that the transmission of a frame must start on a 20 symbol (equals 320 μs in the 2.4 GHz band) backoff slot boundary defined relative to the beacon transmission time.
- The CSMA-CA algorithm requires one (unslotted) or two (slotted) clear channel assessments to be performed one/two backoff slot boundaries before the actual transmission.

- In case of a CCA failure the transmission has to be delayed for a random time period of 0 to 255 backoff periods (equals 0 μs to 5100 μs in the 2.4 GHz band).
- The transmission of an acknowledgements must start between 12 symbols (equals 192 μs in the 2.4 GHz band) and 32 symbols (512 μs in the 2.4 GHz band) after the reception of the last symbol of the previous data or MAC command frame.

On a typical mote platform, these requirements can practically not be met by a platform independent MAC protocol [4], rather they should be pushed from the MAC to the PHY, ideally to hardware. Consequently we believe that the CSMA-CA algorithm and the transmission of acknowledgements should be exposed as services by the radio abstraction, which we consider a part of the execution environment rather than the *TKN15.4* MAC. We present our radio driver interface definitions in Section 4.3.2.

3.1.2 Modularity

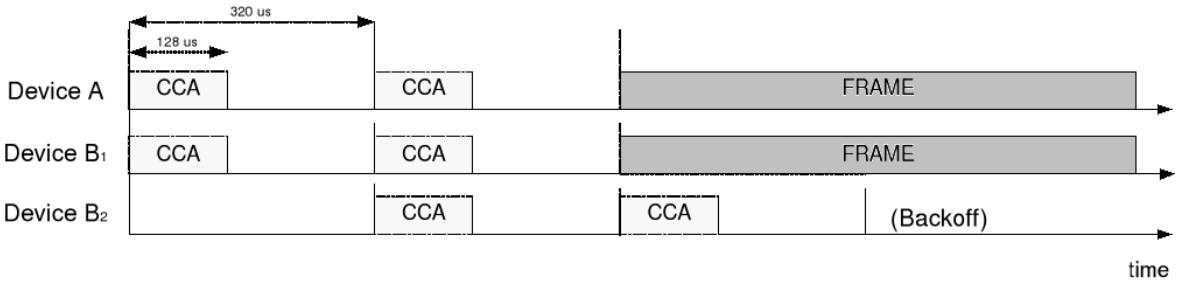
The 802.15.4 MAC supports different channel access methods, network topologies, operating modes and device types. It is thus not surprising that full-blown commercial implementations of the 802.15.4 MAC/PHY result in code sizes of up to 37.3 kB [5]. Given that a typical mote platform has about 48 kB of program memory and 10 kB of RAM [20], an 802.15.4 MAC implementation should follow a modular design that allows to select only the subset of the MAC functionality required by the particular application. *TKN15.4* achieves modularity mainly by mapping the MAC services to software components and allowing to select a suitable subset at compile time. While *TKN15.4* defines some basic composability dependencies and restrictions, it is currently left to the application developer to make reasonable choices about which components are to be in- or excluded. The definition of suitable MAC functionality subsets is not in the scope of this document but a topic of our future work.

3.1.3 Extensibility

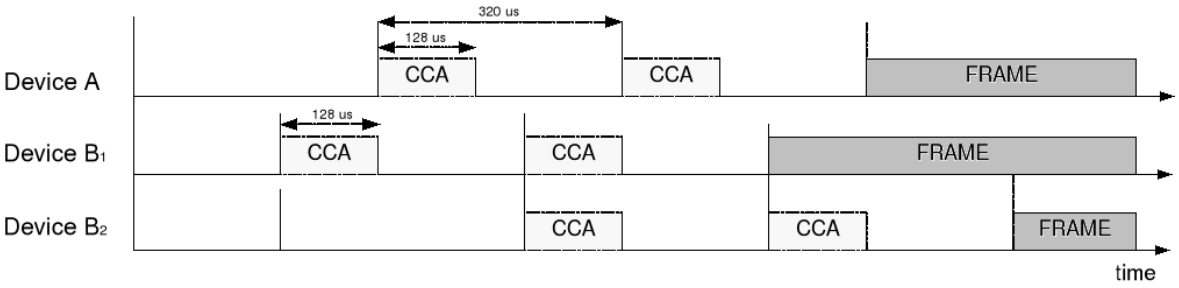
Recently there have been a number of proposals on how the 802.15.4 MAC could be modified to, for example, be able to better adjust the active period in beacon-enabled PANs [19], allow implicit GTS allocation [12] or reduce collisions in multi-hop beacon-enabled PANs [24]. Furthermore, the IEEE 802.15 WPAN Task Group 4e is currently defining a MAC amendment to "enhance and add functionality to the 802.15.4-2006 MAC to better support the industrial markets [...]" [11]. To support the research community in evaluating MAC extensions and facilitate the transition to a future MAC revision the design of *TKN15.4* is kept extensible: the component based design in conjunction with the *TKN15.4* radio arbitration mechanism (explained in Section 4.2) allow flexible integration of new components and/or modification of the superframe structure.

3.2 Challenges

The particular challenges for a TinyOS 802.15.4 MAC implementation and the standard's inherent challenges related to the drift of physical clocks is discussed in the following.



(a) Perfect synchronization of the slot boundaries



(b) Misalignment the slot boundaries by 8 symbols (128 μ).

Figure 3.1: As long as devices are synchronized (a) during the CAP of a beacon-enabled PAN frames will only collide if they are transmitted in the same backoff slots, because otherwise the CCA mechanism will detect a busy channel: the frame from Device A would collide only with the frame from Device B_1 but not with a frame from B_2, because B_2 would detect a busy channel and backoff. However, if due to clock drift devices lose synchronization (b), then their frames can collide with frames that are transmitted in subsequent backoff slots: the frame from Device A would collide with both, the frame from Device B_1 and B_2.

3.2.1 TinyOS 2 Integration

There are two main TinyOS-specific challenges for a platform independent 802.15.4 MAC implementation: first, TinyOS is not a real-time operating system, yet in beacon-enabled mode some operations need to be accurately timed. *TKN15.4* solves this problem by pushing most time-critical operations from the MAC to the chip-specific radio drivers, because the drivers typically operate in a low-latency (interrupt) context and can better exploit the particular hardware characteristics, for example hardware-generated acknowledgements [25].

Second, TinyOS 2 MAC protocols are traditionally radio chip specific: the abstraction of a particular radio chip already includes a certain MAC protocol. Since there exists no platform independent TinyOS 2 MAC protocols and there are no established interfaces or guidelines on how the radio hardware should be exposed or how MAC protocols are to be structured, one contribution of this work is the proposal of a set of platform independent radio interfaces to be provided by a 802.15.4-compliant radio chip abstraction in TinyOS 2 (Section 4.3.2).

3.2.2 Clock Drift

To the best of our knowledge, it has so far not been recognized that clock drift can cause contradictions with the timing requirements of the slotted CSMA-CA in beacon-enabled PANs. As explained in Section 3.1.1, in the CAP the slotted CSMA-CA algorithm requires frames to be transmitted on backoff slot boundaries. In conjunction with the listen-before-send mechanism slotted transmissions theoretically guarantee that the time interval during which the arrival of one packet implies a collision with another packet ("vulnerability period") is limited to one backoff slot (320 μs). This is shown in Figure 3.1a. In practice, however, clock drift can result in collisions between frames even if they are transmitted in subsequent backoff slots and thus the practical vulnerability period is larger. This is explained in the following.

A beacon represents the synchronization point in a superframe: every device must receive the beacon as it determines the slot boundaries within the CAP (cf. Figure 2.1). The active portion of a superframe is bounded by the beacon interval (BI), which is defined as:

$$BI = aBaseSuperframeDuration * 2^{BO}$$

where $0 \leq BO \leq 14$ for beacon-enabled PANs and $aBaseSuperframeDuration = 960$ [symbols], resulting in a maximum beacon interval of about 250 seconds in the 2.4 GHz band. The CAP may span the entire active portion, which means that the slotted CSMA-CA must be applicable until the end of the beacon interval, i.e. for up to 250 seconds after the synchronization point.

The standard requires a system clock with a tolerance as great as $\pm$40 ppm. This means that a device may have a clock deviation of $\pm$8 symbols ($\pm$ 128 μs) compared to the nominal time 3.2 seconds after the beacon, which is possible with beacon order $\geq$ 8. The device would then access the channel 8 symbols before/after the actual backoff slot boundary. One important consequence is that its frame may then collide with a frame from another device which has perfect timing (zero clock drift) even if the two frames are transmitted in different backoff slots. This is depicted in Figure 3.1b.

The effect can manifest when the relative offset between the clocks of two devices is between 8 and 12 symbols, i.e. starting 1.6 seconds after the beacon arrival which is possible in PANs operating with beacon order $\geq$ 7. The practical implications of this effect on the performance of a 802.15.4 network (in terms of throughput, delay, etc.) are not entirely clear, but it is likely that throughput decreases due to an increased collision probability.

Clock drift is a non-negligible property of physical clocks and frequency skew of $\pm$ 40 ppm is a typical value for quartz crystal oscillators. The effects of ambient temperature and aging are responsible for additional clock drift. As stated by authors of the standard themselves: "very small single-chip 802.15.4 implementations could well vary between $\pm$ 40 ppm during a 16-second beacon period ($macBeaconOrder = 10$)" [8]. However, to the best of our knowledge none of the existing MAC performance evaluation studies have taken clock drift into consideration and we consider a re-evaluation of this aspect important future work.

Chapter 4

TKN15.4 Architecture

The *TKN15.4* architecture covers a platform independent 802.15.4 MAC implementation and defines the interfaces towards the layer below (PHY / radio driver) and above (to the network layer). The design and implementation of the radio driver (PHY) is platform/chip specific and thus not part of the *TKN15.4* architecture. This chapter introduces the MAC architecture, explains its radio arbitration mechanism and lists its key interfaces. While the component breakdown is decoupled from a specific operating system in this section we use nesC [7] syntax and refer to existing TinyOS 2 library components and established TinyOS software design patterns [6] whenever applicable.

4.1 MAC Decomposition

Figure 4.1 shows an architectural overview of *TKN15.4*, its main components and the interfaces that are used to exchange MAC frames between components. While this figure abstracts from the majority of interfaces and some configuration components, it illustrates one important aspect, namely how access to the platform specific radio driver (PHY) is structured. For the purpose of explanation the *TKN15.4* MAC can be subdivided into three sublayers:

On the lowest level (dark gray boxes), the `RadioControlP` component manages the access to the radio: with the help of an extended TinyOS 2 arbiter component it controls which of the components on the level above is allowed to access the radio at what point in time. The use of a TinyOS resource arbiter avoids inconsistencies in the radio driver state machine and is in line with the standard TinyOS 2 resource usage model: before a component may access a resource it must first issue a request; once it is signalled the `granted()` event by the arbiter the component can use the resource exclusively; and after usage the resource must be released.[1] *TKN15.4* extends this model by allowing a component that owns the radio resource to dynamically transfer the ownership to a specific other component. This is explained in Section 4.2.

Most of the components on the second level (medium gray boxes) represent different parts of a superframe: the `BeaconTransmitP`/`BeaconSynchronizeP` components are responsible for transmission/reception of the beacon frame, the `DispatchSlottedCsmaP` component manages

[1] `http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep108.html`

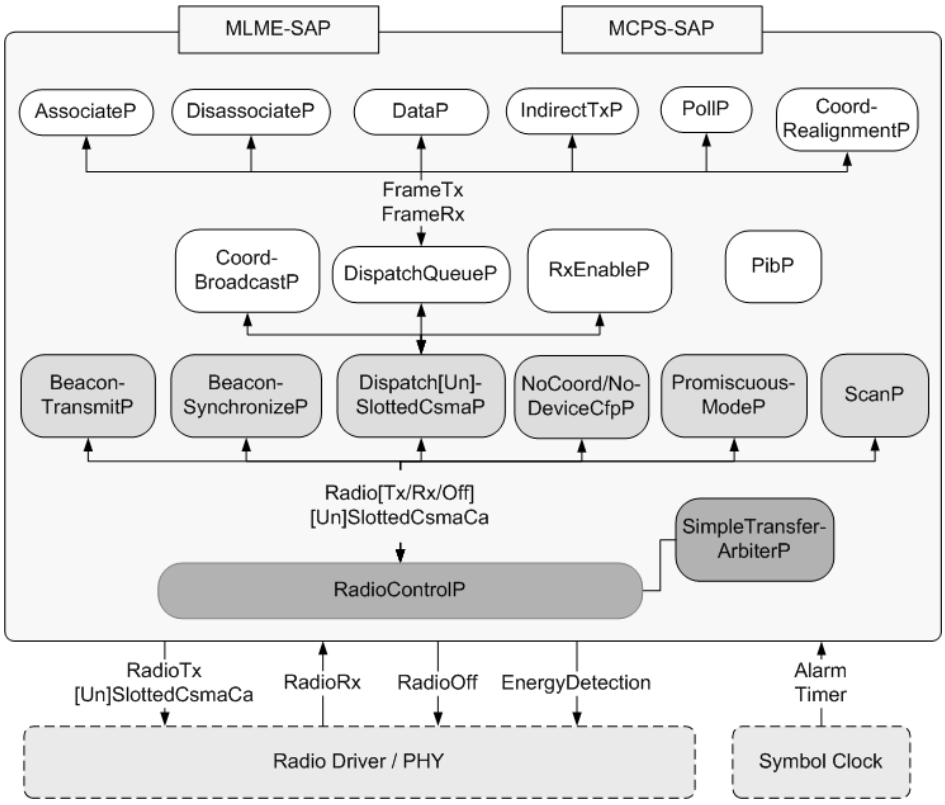


Figure 4.1: The *TKN15.4* architecture: components are represented by rounded boxes, interfaces by connection lines. The radio driver and symbol clock components are external to *TKN15.4*.

Component Name	Function [Provided IEEE 802.15.4 Interface]
`AssociateP`	PAN association [MLME-ASSOCIATE,MLME-COMM-STATUS]
`BeaconTransmitP`	periodic beacon transmission [MLME-START]
`BeaconSynchronizeP`	periodic beacon tracking [MLME-SYNC(-LOSS), MLME-BEACON-NOTIFY]
`CoordBroadcastP`	transmission of coordinator broadcast frames
`CoordRealignmentP`	managing realignment commands [MLME-ORPHAN, MLME-COMM-STATUS]
`DataP`	assembling/dispatching data frames [MCPS-DATA, MCPS-PURGE]
`DisassociateP`	PAN disassociation [MLME-DISASSOCIATE]
`DispatchQueueP`	send queue for data/command frames
`DispatchSlottedCsmaP`	frame transmission/reception during CAP (excluding the CSMA-CA algorithm)
`DispatchUnslottedCsmaP`	frame transmission/reception in nonbeacon-enabled PANs (excluding the CSMA-CA algorithm)
`IndirectTxP`	managing indirect transmissions
`PibP`	maintaining the PAN Information Base [MLME-RESET, MLME-GET, MLME-SET]
`PollP`	requesting data from a coordinator [MLME-POLL]
`PromiscuousModeP`	enabling/disabling promiscuous mode
`RadioClientC`	virtualizing access to `RadioControlP`
`RadioControlImplP`	access control to the radio
`RadioControlP`	configuration for `RadioControlImplP`
`RxEnableP`	enabling the receiver during idle time [MLME-RX-ENABLE]
`ScanP`	channel scanning [MLME-SCAN]
`SimpleTransferArbiterP`	radio resource arbitration
`TKN154BeaconEnabledP`	MAC configuration when used in a beacon-enabled PAN
`TKN154NonBeaconEnabledP`	MAC configuration when used in a nonbeacon-enabled PAN

Table 4.1: The *TKN15.4* MAC components.

frame transmission and reception during the CAP and the `NoCoordCfpP`/`NoDeviceCfpP` components are responsible for the CFP.[2] In nonbeacon-enabled mode a superframe structure is not used and these components are replaced by the `DispatchUnslottedCsmaP` component, which is then responsible for frame transmission and reception in nonbeacon-enabled mode. For reasons stated in Section 3.1.1 *TKN15.4* does not implement the (un)slotted CSMA-CA algorithm: the `DispatchSlottedCsmaP`/`DispatchUnslottedCsmaP` components are responsible for the initialization of the CSMA-CA parameters, but the algorithm is implemented and executed in the platform specific radio driver (Section 4.3.2). In either beacon- or nonbeacon-enabled mode the `ScanP` and `PromiscuousModeP` components are providing services for channel scanning and enabling/disabling promiscuous mode, respectively. The radio arbitration mechanism is used to coordinate the activities of the components on this level so that they do not overlap in time: typically a component is "active" only while it has exclusive access to the radio resource. It then performs a certain task (e.g. transmission of a beacon or performing a channel scan) and afterwards either releases the resource or passes it on to some other component. This is explained in Section 4.2.

The components on the top level (white boxes) implement the remaining MAC data and management services, for example, PAN association or requesting (polling) data from a coordinator. These services typically utilize data and command frame transmission/reception based on the (un)slotted CSMA-CA algorithm and consequently the components are "wired" via a send queue, `DispatchQueueP`, to either `DispatchSlottedCsmaP` (in beacon-enabled PANs) or `DispatchUnslottedCsmaP` (in nonbeacon-enabled PANs). A component on this level typically provides a certain MAC `MLME/MCPS` primitive to the next higher layer, it is

[2] In its current version *TKN15.4* does not include a GTS implementation, the `NoCoordCfpP`/`NoDeviceCfpP` components are empty placeholder component.

responsible for assembling the particular data or command frame and it accepts and processes incoming frames of the same type. For example, the `DataP` component provides the `MCPS-DATA` primitive to the next higher layer to send a frame to a peer device. On receipt of the `MCPS-DATA.request` primitive `DataP` will assemble the data frame and enqueue it in the send queue `DispatchQueueP`. The `Dispatch[Un]SlottedCsmaP` component will eventually dequeue the frame, and manage its transmission. Afterwards it will signal a completion event to the `DataP` component, which in turn propagates a `MCPS-DATA.confirm` event back to the next higher layer including an appropriate status code that denotes whether the transmission was successful or not.

The next higher layer accesses all MAC services either via the `TKN154BeaconEnabledP` component (in beacon-enabled PANs) or via the `TKN154NonBeaconEnabledP` component (in nonbeacon-enabled PANs). These configuration components are nesC *facades* [6] responsible for "wiring" the MAC components together, respectively. They allow to disable/remove certain MAC functionality by specifying empty *placeholder* [6] components. Table 4.1 lists all *TKN15.4* MAC components together with the provided `MCPS/MLME` primitives.

4.2 Radio Arbitration in Beacon-Enabled PANs

TKN15.4 includes a radio arbitration component (`SimpleTransferArbiterP`). With the help of this arbiter the `RadioControlP` component decides which component may access the radio at what time. Resource arbiters are part of the TinyOS 2 library and explained in TEP 108.[3] Two properties make the standard arbiters suboptimal for *TKN15.4* when used in beacon-enabled PANs: (1) the transition of the resource ownership from one component to another may take an arbitrary time, because it typically involves posting a task (the `Resource.granted()` event is signalled in a separate task) and (2) the resource arbitration model is based on the assumption that clients are relatively independent of each other and arbitration is based on a fixed queuing policy (e.g. round robin or first-come-first-serve).

The first is suboptimal, because a superframe is represented by multiple components and just like there is a continuous transition between the different parts of a superframe in time the access to the radio should be transferred immediately between the involved components. For example, on a coordinator node immediately after a beacon has been transmitted the radio should be switched to receive mode. If the component responsible for managing the CAP is granted the access to the radio too late (because there are other tasks served before), then incoming frames may be lost.

The second is suboptimal, because a superframe is dynamic and some parts of it are dependent on others. For example, a beacon defines whether the following CFP is present and missing an incoming beacon means that the entire superframe cannot be used. Consequently the components that represent the superframe should be able to decide cooperatively at what time which of them should be "active". Instead of treating them as independent clients that compete for access to the radio, it is more natural to let them coordinate when and in what order they access the radio. Therefore we extend the standard `Resource` interface by a single command and a single event that enables immediate transfer of the resource from one client to another. The transfer does not involve posting a separate task and may override the

[3] `http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep108.html`

```
interface TransferableResource{
  async command error_t request();
  async command error_t immediateRequest();
  event void granted();
  async command error_t release();
  async command bool isOwner();
  async command error_t transferTo(uint8_t dstClient);
  async event void transferredFrom(uint8_t srcClient);
}
```

Figure 4.2: The `TransferableResource` interface extends the standard `Resource` interface by an additional command `transferTo()` and an event `transferredFrom()` to transfer the ownership of the resource from one client to another.

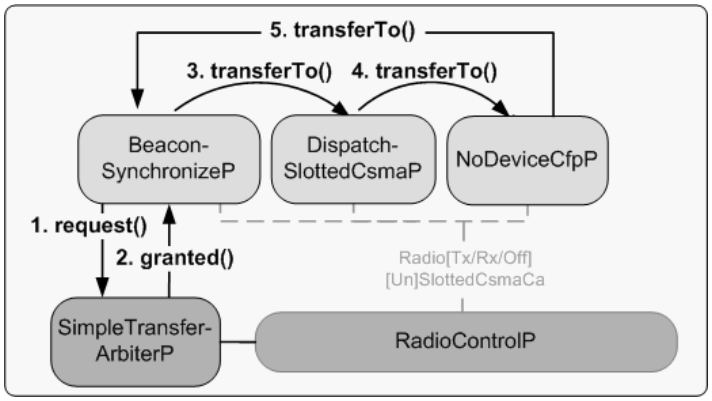


Figure 4.3: Transferring the radio token between the components responsible for an incoming superframe. The commands `request(), transferTo()` and the `granted()` event are part of the `TransferableResource` interface.

default queuing policy. We call this extended interface `TransferableResource`, it is shown in Figure 4.2.

In the following we call the radio resource the "radio token", because a token better matches the notion of transferability. Figure 4.3 shows an example of how the radio token is shared between the components responsible for an incoming superframe: as long as the next higher layer has requested synchronziation with the coordinator, the `BeaconSynchronizeP` component will always have a request pending for the radio token (1). After it has been granted the radio token (2) it will try to track the beacon from the coordinator. Once the beacon has been received, the radio token will immediately be transferred to the `DispatchSlottedCsmaP` component (3), which is then responsible for managing the CAP. When the CAP has finished the radio token is transferred to the component responsible for the CFP (4). Finally, at the end of the CFP the token is passed back to `BeaconSynchronizeP` (5) to be able to receive the next beacon. Afterwards, as long as no other components request the radio token, the steps (3)-(5) repeat indefinitely.

Only the component that owns the radio token may access the radio and whenever a component does not own the token, it is typically "inactive": it may accept requests from the next higher layer, but it will typically have to wait until it is transferred/granted the token

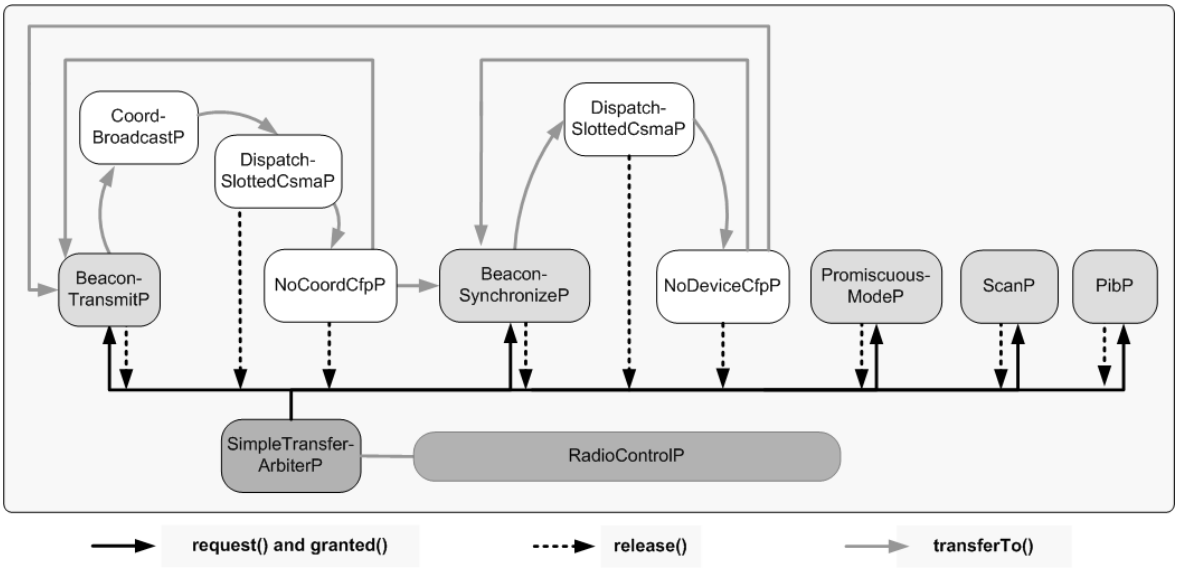


Figure 4.4: Resource arbitration in a beacon-enabled PAN: components that never actively request the radio token are shown in white boxes; components that may request the token are shown in medium gray color. The commands `request(), transferTo(), release()` and the `granted()` event are part of the `TransferableResource` interface.

before it can serve the requests. A component owning the radio token must make sure that it gives up the token at latest when the corresponding part of the superframe has finished (cf. Figure 2.1). For example, whenever the `DispatchSlottedCsmaP` component is given the token from the `BeaconSynchronizeP` component, it will first set an `Alarm` to expire at the end of the CAP (less a platform specific guard time). And when this `Alarm` expires, it will transfer the token to the component responsible for the CFP.

As shown in Figure 4.4 in addition to the `BeaconSynchronizeP` component the `BeaconTransmitP`, `ScanP`, `PipB` and `PromiscuousModeP` components can also request the radio token from the arbiter. Through interfaces not shown here a client of the arbiter can always check, whether other clients have requested the token and then release it. For example, whenever the `PipB` component requests the token to serve a `MLME-RESET` request from the next higher layer, then the components responsible for the superframe will always give up the token via the `TransferableResource.release()` command. When a token is released the arbiter will allocate it to the next client based on a round-robin policy. Figure 4.4 shows all possible transitions of the radio token from one component to another.

In a nonbeacon-enabled PAN a superframe structure is not used and the radio token is never transferred between components. Instead components request it (via `TransferableResource.request()`) and release it (via `TransferableResource.release()`) according to the standard TinyOS resource arbitration policy.[4]

[4] `http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep108.html`

```
interface MCPS_DATA {
   command ieee154_status_t request (message_t *frame, uint8_t payloadLen,
      uint8_t msduHandle, uint8_t txOptions);
   event void confirm ( message_t *frame, uint8_t msduHandle,
      ieee154_status_t status, uint32_t timestamp);
   event message_t* indication ( message_t* frame );
}
interface MLME_BEACON_NOTIFY {
   event message_t* indication ( message_t *beaconFrame );
}
```

Figure 4.5: The `MCPS_DATA` and `MLME_BEACON_NOTIFY` primitives are adapted to support `message_t` and buffer swapping.

4.3 Interface Definitions

The 802.15.4 standard specifies 17 MAC and 6 PHY primitives [15]. *TKN15.4* slightly adapts the MAC primitives to better match TinyOS 2 design principles; this is explained in Section 4.3.1. For reasons stated in Section 3.1.1 the PHY (radio driver) interfaces are revised; these interface definitions are presented in Section 4.3.2.

4.3.1 Interfaces Towards the Next Higher Layer

TinyOS 2 does not support dynamic memory allocation, instead most allocation and binding is pushed to compile time, which makes code more robust and predictable. This also holds for the allocation of message buffers: a component that wants to send a packet is responsible for allocating a `message_t` buffer, which is the TinyOS 2.x message buffer abstraction that is used by protocols on network layer and above (cf. TEP 111[5]). Consequently *TKN15.4* adapts the MAC primitives to support `message_t` and the TinyOS buffer swapping semantics. This is relevant for two MAC primitives, `MCPS-DATA` and `MLME-BEACON-NOTIFY`; Figure 4.5 shows the revised interface definitions.

In contrast to the primitives specified in the 802.15.4 standard, these interfaces do not transport control information (source/destination address of the frame, etc.) explicitly. Rather, this information is already contained in a `message_t` and can be configured/retrieved via suitable accessor functions (via the `IEEE154Frame` interface, not shown here), which is in line with the TinyOS 2 concept of treating `message_t` as abstract data type. A drawback of using `message_t`, however, is that a message buffer always consumes the maximum MAC payload plus header size, which frequently results in suboptimal memory usage. It is part of our future work to design a more flexible message buffer abstraction for TinyOS 2.

All other `MLME/MCPS` primitives are adopted virtually one-to-one from the 802.15.4 specification, with one exception: because it is usually more convenient for the caller the `MLME_GET/SET` primitives consist of single commands (are non-split-phase), and the beacon payload for an outgoing superframe on a coordinator is accessed not via PIB, but through a separate interface (`IEEE154TxBeaconPayload` interface, not shown here).

[5] `http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep111.html`

```
interface RadioRx {
   async command error_t enableRx(uint32_t t0, uint32_t dt);
   async event void enableRxDone();
   async command bool isReceiving();
   event message_t* received(message_t *frame, const ieee154_timestamp_t *timestamp);
}
interface RadioTx {
   async command error_t transmit(ieee154_txframe_t *frame, const ieee154_timestamp_t *t0, uint32_t dt);
   async event void transmitDone(ieee154_txframe_t *frame, const ieee154_timestamp_t *timestamp,
      error_t result);
}
interface RadioOff {
   async command error_t off();
   async event void offDone();
   async command bool isOff();
}
interface UnslottedCsmaCa{
   async command error_t transmit(ieee154_txframe_t *frame, ieee154_csma_t *csma);
   async event void transmitDone(ieee154_txframe_t *frame, ieee154_csma_t *csma, bool ackPendingFlag,
      error_t result);
}
interface SlottedCsmaCa{
   async command error_t transmit(ieee154_txframe_t *frame, ieee154_csma_t *csma,
      const ieee154_timestamp_t *slot0Time, uint32_t dtMax, bool resume, uint16_t remainingBackoff);
   async event void transmitDone(ieee154_txframe_t *frame, ieee154_csma_t *csma,
      bool ackPendingFlag, uint16_t remainingBackoff, error_t result);
}
interface EnergyDetection {
   command error_t start(uint32_t duration);
   event void done(error_t status, int8_t maxEnergyLevel);
}
```

Figure 4.6: The interfaces that a radio driver must provide to the *TKN15.4* MAC.

4.3.2 Interfaces Towards the Radio Driver

TKN15.4 requires the radio driver to essentially provide the six interfaces shown in Figure 4.6. These interfaces push many time-critical operations from the MAC to the radio driver, because the latter can include platform- and chip-specific code and is thus in a better position to meet the tight timing constraints in beacon-enabled PANs (cf. Section 3.1.1).

The `RadioRx, RadioTx` interfaces enable receive mode and allow the MAC to transmit a frame at a specific point in time, respectively. The `RadioOff` interface allows to disable the transceiver and the `SlottedCsmaCa, UnslottedCsmaCa` interfaces transmit a frame based on the (un)slotted CSMA-CA algorithm. The MAC specifies the initial CSMA-CA parameters but the algorithm itself is implemented and executed in the radio driver. Also, the radio driver is responsible for performing any random backoffs and for the transmission of acknowledgements. The `EnergyDetection` interface is similar to the one defined in the standard. It allows to obtain the maximum energy on a given channel over a certain time period.

Chapter 5

TinyOS-2.x Implementation

This chapter summarizes the status of the implementation, describes the directory structure of *TKN15.4* in the TinyOS-2.1 module, explains the steps necessary for making *TKN15.4* available on a new TinyOS 2 platform and introduces the built-in debugging support.

5.1 Implementation Status

At the time of writing this document the *TKN15.4* MAC implementation includes almost the complete functionality described in the 802.15.4-2006 specification, except for (1) GTS allocation and management, (2) security services, and (3) a few minor services like PAN ID conflict notification.[1] *TKN15.4* was developed on and is available for the TelosB/Tmote Sky platform [20]. Reportedly, it has also been successfully ported to the micaZ [2] platform with little effort. While we have successfully performed a few interoperability tests with the certified Texas Instruments 802.15.4 TIMAC [26], we defer the evaluation of *TKN15.4* to a future report.

5.2 Directory Structure

Within the TinyOS 2 module the *TKN15.4* MAC implementation is located in the `tinyos-2.x/tos/lib/mac/tkn154` directory. This directory includes additional subdirectories for the interface definitions and the placeholder components. The location of platform specific (e.g. CC2420 radio driver) code is out of the scope of this document but should follow established TinyOS 2 guidelines.[2] Table 5.1 lists the directories that contain the *TKN15.4* MAC implementation and, as an example, the directories containing the platform specific code for the TelosB [20] platform.

5.3 Platform Requirements

To make the *TKN15.4* MAC available on a new TinyOS 2 mote platform essentially the following three requirements have to be met:

[1] For the details please refer to `tinyos-2.x/tos/lib/mac/tkn154/README.txt`

[2] http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/README

Content	Directory in the TinyOS-2.1 Tree
TKN15.4 MAC implementation	
components	`tinyos-2.x/tos/lib/mac/tkn154`
interfaces	`tinyos-2.x/tos/lib/mac/tkn154/interfaces`
placeholder components	`tinyos-2.x/tos/lib/mac/tkn154/dummies`
test applications	`tinyos-2.x/apps/tests/tkn154`
Platform specific code for TelosB	
configuration files	`tinyos-2.x/tos/platforms/telosb/mac/tkn154`
modified timer subsystem	`tinyos-2.x/tos/platforms/telosb/mac/tkn154/timer`
CC2420 radio driver	`tinyos-2.x/tos/chips/cc2420_tkn154`

Table 5.1: *TKN15.4* directories in the TinyOS-2.1 tree.

1. The platform must include a radio driver that provides the interfaces listed in Section 4.3.2. When a platform supports only the nonbeacon-enabled mode then the radio driver does not have to provide the `SlottedCsmaCa` interface, if it supports only the beacon-enabled mode, then the `UnslottedCsmaCa` interface is not required. In addition, the radio driver must use the TinyOS 2 `Notify` interface to be updated about changes in the PHY PIB, and provide a set of PHY-specific constants.[3]

2. The MAC implementation uses `Alarm` and `Timer` interfaces with a precision of a 802.15.4 symbol (62.5 kHz for the 2.4 GHz band) and an accuracy of ±40 ppm. The `Alarm` and `Timer` interfaces are standard TinyOS 2 interfaces that are provided by a platform's timer subsystem.[4] However, a TinyOS 2 platform typically does not provide these interfaces with the required precision/accuracy. In this case the platform's timer subsystem must be extended accordingly (possibly including the use of additional hardware).[5]

3. The platform must provide a TinyOS configuration component that connects ("wires") the platform independent *TKN15.4* MAC implementation `TKN154BeaconEnabledP` or `TKN154NonBeaconEnabledP` to the platform specific timer subsystem and radio driver. In addition, the platform has to define a set of guard time constants.[6]

5.4 Example Applications and Debugging Support

At the time writing this document there exist 5 example application for the beacon-enabled mode. They are located in the `tinyos-2.x/apps/tests/tkn154` folder and can be used to test PAN association/disassociation, periodic beacon transmission/reception, direct and indirect data transmission and the promiscuous mode. The applications have been successfully tested on the TelosB platform [20] (and reportedly on the micaZ [2] platform). The

[3] for an example see `tinyos-2.x/tos/chips/cc2420_tkn154/CC2420TKN154C.nc` and `TKN154_PHY.h`

[4] `http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep102.html`

[5] for an example see `tinyos-2.x/tos/platforms/telosb/mac/tkn154/timer`

[6] for an example see `tinyos-2.x/tos/platforms/telosb/mac/tkn154/Ieee802154BeaconEnabledC` and `tinyos-2.x/tos/platforms/telosb/mac/tkn154/TKN154_platform.h`

```
$ cd tinyos-2.x/apps/tests/tkn154/TestStartSync/coordinator
$ TKN154_DEBUG=1 make telosb install
...
$ java net.tinyos.tools.PrintfClient -comm serial@/dev/ttyUSB0:115200
PibP:181:MLME_RESET.request(1) -> result:  0
BeaconTransmitP:247:MLME_START.request -> result:  0
BeaconSynchronizeP:381:Token granted.
BeaconTransmitP:389:Putting new superframe spec into operation
BeaconTransmitP:436:Sending a beacon immediately.
BeaconTransmitP:501:Beacon Tx scheduled
BeaconTransmitP:516:Beacon Tx success at 1242
DispatchSlottedCsmaP:198:Got token, remaining CAP time:  30222
DispatchSlottedCsmaP:526:CapEndAlarm.fired()
DispatchSlottedCsmaP:169:updateState() transitions:  7711
DispatchSlottedCsmaP:370:Handing over to CFP.
BeaconSynchronizeP:407:Token transferred, will Tx beacon in 332
BeaconTransmitP:501:Beacon Tx scheduled for 31962.
BeaconTransmitP:516:Beacon Tx success at 31962
DispatchSlottedCsmaP:198:Got token, remaining CAP time:  30214
...
```

Figure 5.1: The debugging output for one of the *TKN15.4* example applications. The second line installs the `TestStartSync` test application with debugging support on a TelosB node. The next line starts the TinyOS 2 PrintfClient application (see `tinyos-2.x/apps/tests/TestPrintf/`). The following output shows the debugging information. For every `dbg_serial()` statement in the source code there is one line of debugging output starting with the component name, followed by the line number in the source code and the debugging string.

nonbeacon-enabled mode has only been cursorily tested and no test applications are available yet.

Debugging embedded systems is usually complicated and time consuming. *TKN15.4* supports debugging over the serial interface (USB) using printf()-like debugging and assertions. The first is exposed through the `dbg_serial()` macro, the second through the `ASSERT()` macro, both defined and explained in `tinyos-2.x/tos/lib/mac/tkn154/TKN154_MAC.h`. Figure 5.1 shows the debugging output for one of the example applications.

Chapter 6

Related Work

There exist a few open-source implementations of the 802.15.4 MAC: Meshnetics [17] provides an open-source 802.15.4 MAC implementation for TinyOS 1.x that was developed for Atmel-based platforms. It covers only the nonbeacon-enabled mode (without security services) and is published under the Common Development and Distribution License. Cunha et al. [3] have developed an 802.15.4 MAC implementation as part of their open-source "open-ZB" ZigBee stack in TinyOS 2. The code has been made available only recently, for the most part the development of *TKN15.4* and the open-ZB stack happened in parallel (unnoticed). In contrast to *TKN15.4* open-ZB builds on a monolithic architecture, where the entire MAC is contained in a single component. Finally, J. Flora has developed an open-source 802.15.4 MAC for TinyOS 1.x [4]; the development was made on platform based on a Freescale MC13192 radio and a Motorola HCS08 MCU.

There exist several commercial implementations of the 802.15.4 MAC. For example, as part of their ZigBee "BeeStack" Freescale offers a fully compliant IEEE 802.15.4 MAC including GTS and AES-128 security [5]. The object code is free and usable on different Freescale platforms. The Texas Instruments 802.15.4 TIMAC [26] is available as object code free of charge for the CC2430 platforms and for platforms based on a MSP430 MCU and a CC2420/CC2520 radio. In the latest revision 1.2.1 security services and GTS are not supported. The TIMAC is also used by the Texas Instruments ZigBee stack "Z-Stack". Atmel provides a 802.15.4 MAC [1] for platforms based on an Atmel AT86RF230/RF231 radio. This implementation is available as source code but must only be used on Atmel-based platforms. The latest release (2.1.1) does not include GTS.

Chapter 7

Conclusion

In this document we presented *TKN15.4*, a platform independent IEEE 802.15.4-2006 MAC implementation for the 2.1 release of the TinyOS operating system. We illustrated how *TKN15.4* achieves modularity through a fine-grained component breakdown and how the architecture builds upon an extension of the TinyOS 2 resource arbitration mechanism and a set of revised PHY/MAC interfaces. We showed how the resource arbitration extension enables immediate transfer of the radio resource between the MAC components and thus better matches the continuous transition between the different parts of an 802.15.4 superframe. The resource arbitration mechanism also makes the architecture extensible, because it supports the integration of new components and thus modification of the existing superframe structure. In order to better meet the tight timing constraints in beacon-enabled PANs the PHY interfaces were revised: time critical tasks, such as the CSMA-CA algorithm and the transmission of acknowledgements, are executed in a platform specific radio driver, because it typically operates in a low-latency (interrupt) context and can better exploit the particular hardware characteristics. Consequently, as long as a small set of well-defined requirements are met the MAC implementation can be used on any TinyOS 2 platform – and reportedly, *TKN15.4* has been successfully ported to the micaZ [2] platform with little effort. An evaluation of *TKN15.4*, including interoperability tests with certified MAC implementations, is still missing and will be part of future document.

Bibliography

[1] Atmel. IEEE 802.15.4 MAC User Guide. `http://www.atmel.com/dyn/resources/prod_documents/doc5182.pdf`.

[2] Crossbow Technology Inc. MICAz wireless measurement system. `http://www.xbow.com`, June 2004.

[3] A. Cunha, A. Koubaa, R. Severino, and M. Alves. Open-ZB: an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on TinyOS. In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, pages 1–12, Oct. 2007.

[4] Jan Flora and Philippe Bonnet. Never Mind the Standard Here is the TinyOS 802.15.4 Stack. Technical Report 06-10, University of Copenhagen, 2006.

[5] Freescale. 802.15.4 MAC PHY Software Reference Manual. `http://www.freescale.com/files/rf_if/doc/ref_manual/802154MPSRM.pdf`.

[6] David Gay, Phil Levis, and David Culler. Software design patterns for TinyOS. In *LCTES '05: Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 40–49, New York, NY, USA, 2005. ACM.

[7] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, New York, NY, USA, 2003. ACM Press.

[8] Jose A. Gutierrez, Edgar H. Callaway, and Raymond Barrett. *IEEE 802.15.4 Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensor Networks*. IEEE Standards Office, New York, NY, USA, 2007.

[9] HART Communication Foundation. WirelessHART. `http://www.hartcomm2.org`.

[10] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proc. of the ninth international conference on Architectural support for programming languages and operating systems (ASPL 2000)*, 2000.

[11] IEEE 802.15 WPAN Task Group 4e (TG4e). WPAN Homepage. http://www.ieee802.org/15/pub/TG4e.html.

[12] Anis Koubaa, Mario Alves, and Eduardo Tovar. i-GAME: An Implicit GTS Allocation Mechanism in IEEE 802.15.4 for Time-Sensitive Wireless Sensor Networks. *Real-Time Systems, Euromicro Conference on*, 0:183–192, 2006.

[13] Schumacher Kushalnagar, Montenegro. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement and Goals, 2007.

[14] LAN/MAN Standards Committee of the IEEE Computer Society. *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*, October 2003.

[15] LAN/MAN Standards Committee of the IEEE Computer Society. *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*, September 2006.

[16] P. Levis, D. Gay, V. Handziski, J.-H.Hauer, B.Greenstein, M.Turon, J.Hui, K.Klues, C.Sharp, R.Szewczyk, J.Polastre, P.Buonadonna, L.Nachman, G.Tolle, D.Culler, and A.Wolisz. T2: A Second Generation OS For Embedded Sensor Networks. Technical Report TKN-05-007, Telecommunication Networks Group, Technische Universität Berlin, November 2005.

[17] MeshNetics. OpenMAC. `http://www.meshnetics.com/opensource/mac/`.

[18] Emiliano Miluzzo, Xiao Zheng, Kristof Fodor, and Andrew T. Campbell. Radio characterization of 802.15.4 and its impact on the design of mobile sensor networks. In *Proc. of Fifth European Conference on Wireless Sensor Networks (EWSN 2008)*, 2008.

[19] Zeeshan Hameed Mir, Changsu Suh, and Young-Bae Ko. Performance Improvement of IEEE 802.15.4 Beacon-Enabled WPAN with Superframe Adaptation Via Traffic Indication. In Ian F. Akyildiz, Raghupathy Sivakumar, Eylem Ekici, Jaudelice Cavalcante de Oliveira, and Janise McNair, editors, *Networking*, volume 4479 of *Lecture Notes in Computer Science*, pages 1169–1172. Springer, 2007.

[20] Moteiv Corporation. Tmote Sky Datasheet. `http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf`.

[21] T.R. Park, T.H. Kim, J.Y. Choi, S. Choi, and W.H. Kwon. Throughput and energy consumption analysis of ieee 802.15.4 slotted csma/ca. *Electronics Letters*, 41(18):1017–1019, Sept. 2005.

[22] Iyappan Ramachandran, Arindam K. Das, and Sumit Roy. Analysis of the contention access period of IEEE 802.15.4 MAC. *ACM Trans. Sen. Netw.*, 3(1):4, 2007.

[23] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. Understanding the causes of packet delivery success and failure in dense wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 419–420, New York, NY, USA, 2006. ACM.

[24] Maoheng Sun, Kaijian Sun, and Youmin Zou. Analysis and Improvement for 802.15.4 Multi-hop Network. *Communications and Mobile Computing, 2009. CMC '09. WRI International Conference on*, 2:52–56, Jan. 2009.

[25] Texas Instruments. CC2420 data sheet. `http://focus.ti.com/lit/ds/symlink/cc2420.pdf`.

[26] Texas Instruments. TIMAC - IEEE802.15.4 Medium Access Control (MAC) Software Stack. `http://focus.ti.com/docs/toolsw/folders/print/timac.html`.

[27] V.Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, and D. Culler. Flexible Hardware Abstraction for Wireless Sensor Networks. In *Proc. of 2nd European Workshop on Wireless Sensor Networks (EWSN 2005)*, Istanbul, Turkey, February 2005.

[28] J. Werb, M. Newman, V. Berry, S. Lamb, D. Sexton, and M. Lapinski. Improved quality of service in ieee 802.15.4 mesh networks. *Proceedings of International Workshop on Wireless and Industrial Automation*, pages 1–4, 2005.

[29] Jianliang Zheng and Myung J.Lee. *A Comprehensive Performance Study of IEEE 802.15.4*. IEEE Press Book, 2004.

[30] ZigBee Standards Organization. *ZigBee Specification.* `http://www.zigbee.org`, April 2005.