BiPS – A Real-time-capable Protocol Framework for Wireless Sensor Networks

Dennis Christmann, Tobias Braun, Markus Engel, and Reinhard Gotzhein
Computer Science, University of Kaiserslautern, 67653, Kaiserslautern, Germany

Keywords: Real-time, Wireless Networks, Communication Protocols, Implementations, Operating Systems.

Abstract: Distributed real-time systems present a particular challenge, because two key problems have to be solved during their development: First, deployed protocols must provide deterministic behavior to enable a predictable outcome. Second, the implementations of the protocols have to be in compliance with the stringent timing constraints stated by the protocols to ensure that their runtime behavior remains deterministic. This, particularly, requires an adequate isolation of time-critical protocols from less preferential applications installed on the same node. In this paper, we present the protocol framework BiPS, which tackles these challenges for wireless sensor networks and Imote 2 hardware platforms. Besides summarizing the various MAC protocols – both best effort and real-time-capable protocols – and operating system functionalities provided by BiPS, this paper presents comparative evaluations with TinyOS, a state-of-the-practice operating system for wireless sensor networks, and the real-time operating system RIOT. The results show that protocols realized with BiPS outperforms these solutions w.r.t. predictability of execution times, thereby providing evidence of the advantages of BiPS for real-time systems.

1 INTRODUCTION

In a real-time system, correctness does not only depend on logically correct results, but also on the point in time when the results are produced (Kopetz, 1997). In distributed applications where the system is composed of several interacting nodes, this fundamental requirement does not only make demands on the single nodes and their implementations, but also on the communication system, which has to provide reliable and delay-constrained data transfer. To meet both demands, the realization of distributed real-time systems should be a holistic approach, where protocols are designed w.r.t. the real-time properties they have to fulfill as well as implemented in compliance with required time constraints and in harmony with less time-critical components. This, in turn, makes demands on the used operating systems (OSs), which are responsible for the compliance with local deadlines.

In this paper, we present the *Black burst-Integrated Protocol Stack* (BiPS), a real-time-capable protocol framework for wireless networks, which is used in an industrial project, in which sensor values of a production plant have to be sent over a wireless communication medium with bounded delay. The various protocols provided by BiPS are activated in so-called virtual slot regions, which enable the usage of best effort and real-time-capable protocols within one scenario. To guarantee compliance with time constraints stated by the protocols, the framework runs on top of the hardware – in our case, the sensor platform Imote 2 (MEMSIC Inc., 2013) – without further OS. As consequence, all functionalities of a typical OS – like application schedulers (time- or event-driven) and hardware drivers – are provided by BiPS. Thus, BiPS does not only consist of a set of protocols but also represents a light-weight, tailored, and layered OS (Farooq and Kunz, 2011). It comes with flexible application interfaces to enable abstraction and temporal decoupling between the time-critical protocols of BiPS and higher layers with less stringent time constraints.

Compared to state-of-the-practice protocol implementations, BiPS differs in two points: First, BiPS does not include a single protocol but consists of various MAC (Medium Access Control) protocols, abstract interfaces to integrate new ones, and an architecture to apply various protocols in one scenario. Thereby, BiPS is much more flexible than existing communication solutions like IEEE 802.11 or IEEE 802.15.4, where the combination of several MAC schemes is not possible or severely re-

Christmann, D., Braun, T., Engel, M. and Gotzhein, R.
BiPS – A Real-time-capable Protocol Framework for Wireless Sensor Networks.
DOI: 10.5220/0005938300170027
In *Proceedings of the 6th International Joint Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2016)*, pages 17-27
ISBN: 978-989-758-195-3

stricted. Second, in common implementations, protocols are realized as components of a generic (real-time) OS like TinyOS (e.g., IEEE 802.15.4 (Cunha et al., 2007; Hauer, 2009) or 6LoWPAN (Harvan and Schönwälder, 2008)), Contiki (e.g., IEEE 802.11 (Glaropoulos et al., 2014) or LOADng (Elyengui et al., 2015)), or Nano-RK (e.g., WiDom (Pereira et al., 2007)), whereas BiPS is protocol-centric, i.e., the OS functionalities have been built as auxiliary services around the communication protocols. Hence, BiPS is highly-optimized for the execution of real-time protocols. Since compared to generic OSs, BiPS has some limitations like missing support of synchronization mechanisms (e.g., semaphores), the real-time OS FreeRTOS has been integrated into BiPS in terms of a subordinated application scheduler. Thereby, multi-threading and priority-based preemptive scheduling is supported for higher layers like routing protocols or applications, while the execution of time-critical components remains under control of the BiPS core.

Different from our previous work on BiPS (Braun et al., 2014a), in which the focus was on the interfacing of implementations derived from SDL (Specification and Description Language, (ITU-T, 2012)) models, this paper presents core functionalities of BiPS w.r.t. its application in real-time systems. In this regard, the paper gives special attention to implementation aspects, predictability at runtime, and compliance with timing constraints.

The paper is structured as follows: Section 2 outlines BiPS. Details of the framework and integrated protocols are presented in Sect. 3, and OS functionalities together with implementation details in Sect. 4. In Sect. 5, results of a comparison study between BiPS, TinyOS, and RIOT are presented. Section 6 provides a survey on related work. Finally, Sect. 7 concludes the paper.

2 AN OUTLINE OF BiPS

BiPS is a protocol framework for wireless sensor nodes tailored for the Imote 2 platform (MEMSIC Inc., 2013). This platform is based on an Intel XScale PXA271 controller with 256 KiB SRAM, 32 MiB SDRAM, and 32 MiB FLASH. It supports clock rates up to 416 MHz and integrates the widely used IEEE 802.15.4-compliant CC2420 transceiver (TI, 2013).

The architecture of BiPS is presented in Fig. 1. Besides included protocols like the synchronization protocol *Black Burst Synchronization* (BBS; (Gotzhein and Kuhn, 2011)) and several MAC protocols, the figure additionally shows the communication and application scheduler of BiPS and the application interface to decouple time-critical protocol functionalities from higher-layer protocols.

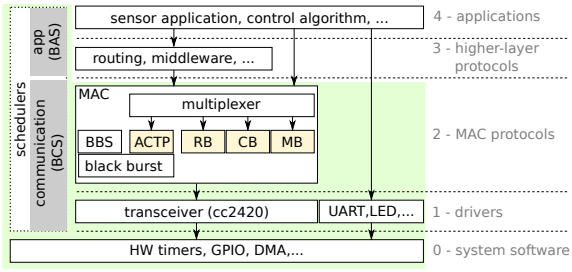

Figure 1: Architecture of BiPS.

The overall structure of BiPS follows a layered approach, where higher layers can abstract from the realization of lower layers by providing abstract interfaces: In layer 0, BiPS implements low level functionality to interact with the hardware, such as access to hardware timers. Layer 1 comprises hardware drivers for peripheral devices, such as an optimized driver for the CC 2420 transceiver using DMA (Direct Memory Access). While layer 0 and 1 are hardware-specific by nature, the protocols in layer 2 abstract from hardware details in most instances. However, full abstraction is not possible, because some limitations of the hardware – e.g., transfer rates and switching delays of the transceiver – must be considered. Currently, layer 2 includes the synchronization protocol BBS providing network-wide synchronization in multi-hop networks and four MAC protocols (see Sect. 3.1). To access the MAC protocols from higher-layer protocols (layer 3) or applications (layer 4), a multiplexer has been introduced, providing a unique interface to all MAC protocols and decoupling time-critical protocols from less time-critical applications (see Sect. 3.3).

The execution of all components – MAC protocols on layer 2 as well as higher-layer protocols and applications on layer 3 and 4 – are controlled by two schedulers: The *BiPS Communication Scheduler* (BCS), developed and optimized for the execution of time-critical MAC protocols, and the *BiPS Application Scheduler* (BAS) for less time-stringent applications. More details on this topic are given in Sect. 4.

3 BiPS AS PROTOCOL FRAMEWORK

The objective of BiPS is to provide an extensible framework for (real-time-capable) MAC protocols and homogeneous and easy-to-use interfaces to access them. In this section, we describe already in-

corporated MAC protocols (Sect. 3.1), mechanisms to run them in parallel in a compatible way (Sect. 3.2), and the multiplexer providing the upper interface to them (Sect. 3.3).

3.1 Available MAC Protocols

Currently, BiPS supports four MAC protocols (see Fig.1): One best effort protocol and three real-time-capable protocols.

The best effort protocol is called *Contention-Based* protocol (short: CB) and realizes a classical CSMA/CA-based (*Carrier Sense Multiple Access with Collision Avoidance*) communication solution. Since backoff times are calculated by means of configurable contention windows, CB allows priority-based frame transmissions and slightly supports Quality-of-Service (QoS), yet without guarantees w.r.t. collisions and delays.

The real-time-capable MAC protocols are called *Reservation-Based* Protocol (short: RB), *Arbitrating and Cooperative Transfer Protocol* (short: ACTP, (Christmann et al., 2012)), and *Mode-Based Scheduling with Fast Mode-Signaling* (short: MB, (Braun et al., 2014b)). RB is a TDMA-based (Time Division Multiple Access) protocol with exclusive slot reservations, hence enabling deterministic guarantees. ACTP is a so-called binary countdown protocol[1] for multi-hop networks, which provides deterministic priority-based medium arbitration with configurable arbitration radius and (network-wide) transmissions of bit sequences within guaranteed delay bounds. For this purpose, it incorporates a collision-resistant communication primitive called *black burst*, which realizes dominant bits by the transmission of a busy tone and recessive bits by the absence of transmission. MB allows, different from RB, a well-controlled amount of dynamic contention for time slots, while still supporting deterministic guarantees and, therefore, enables an efficient usage of the available network resources especially for sporadic messages. To establish the synchronization required by the protocols, BiPS incorporates the synchronization protocol BBS (Gotzhein and Kuhn, 2011), which is designed for dense multi-hop networks and provides network-wide synchronization with low and bounded offset.

Selecting the *best MAC protocol* depends on a scenario's concrete message characteristics and requirements regarding timing and determinism: If, for instance, messages are strictly periodic, RB is the best choice w.r.t. bandwidth usage and absence of collisions.

[1] A famous wired representative of this protocol class is CAN – the Controller Area Network (ISO, 2004).

3.2 Medium Slotting and Virtual Slot Regions

Since large distributed systems realize various applications sending messages with different communication requirements (periodic/sporadic messages with/without timing requirements), it is often not sufficient to run a single MAC protocol only. Instead, the communication system should provide suitable MAC protocols for each application and corresponding interfaces to use them.

In BiPS, this is solved by the possibility to combine all MAC protocols in one scenario by multiplexing their activation in time. For this purpose, BiPS divides time into periodically repeated *super slots* consisting of a configurable number of *macro slots*. Macro slots start with a synchronization phase[2] and have a configurable length, determining the resynchronization interval. Macro slots are further subdivided into so-called *virtual slot regions* with configurable and node-specific positions and durations. Each virtual slot region is self-contained and assigned to one of the MAC protocols, which organizes the medium access within the slot region. The BCS (see Sect. 4.1) is responsible to activate the assigned MAC protocol based on the node's local (but synchronized) clock and to prevent interference between neighboring slot regions. In contrast to the periodically repeated super slot, each macro slot of the super slot may have an individual structure of virtual slot regions with different lengths and assigned protocols.

Figure 2 shows an example of a super slot configuration for three nodes. In the example, a super slot consists of three macro slots with different configurations of virtual slot regions for each of the three nodes. The slot regions have been assigned to different MAC protocols, according to the scenario's communication characteristics. The access to a virtual slot region – and, hence, to a particular MAC protocol – is managed by so-called *Transmission Opportunities* (TOs), which are responsible for the (de-)multiplexing of transmissions (see Sect. 3.3).

The configurable length of slot regions as well as the flexible assignment to MAC protocols enables tailoring of the communication schedule to a specific scenario. Unassigned time slices of macro slots are automatically joined to idle regions (white areas between virtual slot regions in Fig. 2). Since a node

[2] We apply BBS but actually, any synchronization protocol realizing network-wide synchronization with bounded offset and convergence delay could be used.

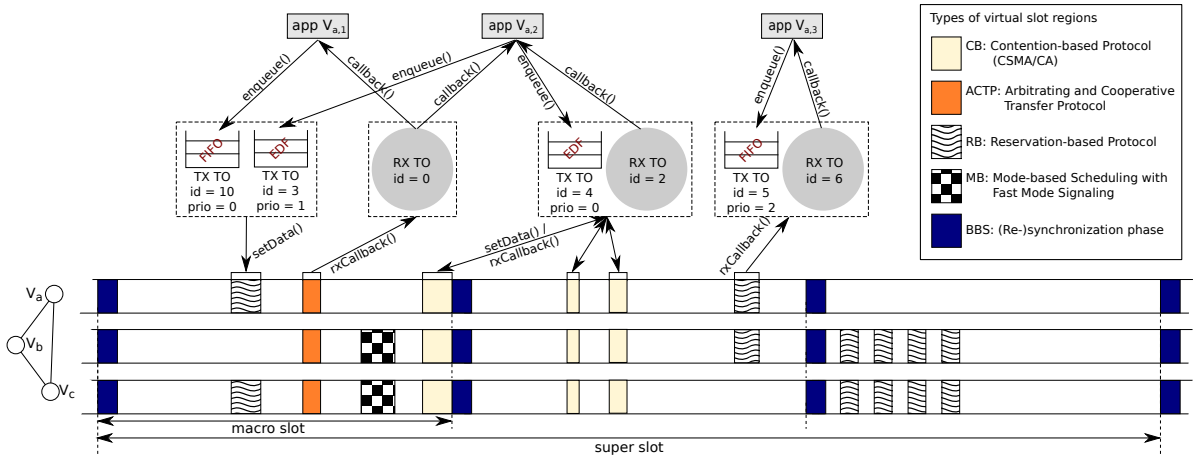

Figure 2: Configuration of virtual slot regions of three nodes and associated transmission opportunities.

is not involved in communication during idle regions, the BCS automatically disables the transceiver to save energy. Therefore, BiPS inherently incorporates a transparent and application-independent duty cycling model on communication level, which is a precondition for wireless sensor networks (WSNs).

3.3 Multiplexing of Applications and Access to MAC Protocols

In BiPS, applications and higher-layer protocols (layers 4 and 3) do not directly interact with the MAC protocols but via TOs, which are located in the multiplexer (see Fig. 1). This has several reasons: First, MAC protocols have time-critical functionalities, whereas applications and higher-layer protocols have less requirements. Second, a direct interaction with MAC protocols would require the protocols to implement queueing facilities, which would increase the complexity of the protocol. Furthermore, directly interacting with a MAC protocol would complicate the exchange of the protocol if a different protocol seems to be more suitable for an application later.

To achieve that the usage of a MAC protocol is, from an application's perspective, transparent, easy, and independent from the concrete protocol, BiPS introduces the multiplexer, realizing a temporal decoupling and loose coupling between MAC protocols and components on higher layers. This multiplexer provides a homogeneous interface to all MAC protocols, though properties of a single transmission may differ.

The multiplexer comprises a set of TX queues – called TX TOs – to store outgoing transmissions until they are ready to be sent. If an application or higher-layer protocol intends to transmit a frame, it enqueues the frame in a TX TO, which is associated with a set of virtual slot regions and owns a priority. When a new virtual slot region begins, the first frame of the associated TX TO with highest priority is transferred to the MAC protocol responsible for this slot region. This protocol then tries to send the frame and informs the multiplexer about success or failure afterwards, which, in turn, informs the initiator of the transmission or – depending on a configuration – triggers retransmissions. Further configuration options of TX TOs allow a limit on the queue size and the selection of a queueing strategy, where currently First-In-First-Out (FIFO) and Earliest-Deadline-First (EDF) are supported. Since a TX TO implements queueing facilities and multiple applications can share a TX TO, the same MAC protocol can be used by several applications executed on the same node. In this case, the TX TO also guarantees the correct delivery of transmission confirmations.

Similar to TX TOs, the multiplexer provides RX TOs, where applications can register a function callback, which is invoked whenever data is received in an associated slot region. In this regard, multiple callbacks can be registered in one RX TO to inform several applications about the reception of a data frame.

The interplay of TX/RX TOs and applications, and the association of TOs to virtual slot regions is illustrated in Fig. 2. In this example, three applications installed on node V_a access the multiplexer of BiPS, which comprises four TX TOs and three RX TOs, each identified with a unique identifier. The example, particularly, highlights the following properties: First, one application can use several TX and RX TOs. Second, several TX TOs can be assigned to one virtual slot region, where priorities enforce a local preference order. Third, TX TOs can be associated with several slot regions to give a frame several transmission chances per super slot. A similar association holds for RX TOs.

4 OPERATING SYSTEM FUNCTIONALITIES IN BiPS

To run real-time-capable protocols, time-critical protocol functionality must be executed in time and with minimal delays. Therefore, BiPS is specially tailored to its target platform Imote 2 and implemented as bare-metal solution. Instead of extending an OS by protocols, which is actually state-of-the-practice (see e.g, (Basmer et al., 2011)), BiPS provides a subset of OS functions while simultaneously focusing on the needs of real-time-capable protocols. Hence, BiPS combines handcrafted optimized code for the Imote 2 platform (e.g., an interrupt handler written in assembler and optimized hardware drivers using DMA) with abstract interfaces and schedulers for MAC protocols as well as less time-critical applications.

According to the design of BiPS, we distinguish between core functionalities with high requirements w.r.t. timeliness and components with lower requirements. To reflect this, BiPS incorporates two schedulers: The *BiPS Communication Scheduler* (BCS, Sect. 4.1) responsible for the execution of time-critical components (MAC protocols, core functions of BiPS), and the *BiPS Application Scheduler* (BAS, Sect. 4.2) executing applications and higher-layer protocols. Both benefit from core services regarding memory management and fault handling (Sect. 4.3).

4.1 The BiPS Communication Scheduler

The main task of the BCS is the timely (de-)activation of MAC protocols at the start and end of associated virtual slot regions, respectively. In order to prioritize BCS over BAS, BCS and its managed components run in the interrupt mode of the microcontroller. To ensure timeliness when (de-)activating MAC protocols, the BCS uses a dedicated hardware timer of the Imote 2 platform.

Execution of MAC protocols is controlled by the BCS via a uniform interface, which has to be implemented by each MAC protocol and prescribes functions to be implemented to exchange data with the multiplexer. Thereby, the extension of the framework with a new protocol is straightforward. Virtual slot regions are structured by the BCS in a protocol-independent way, which ensures the isolation of neighboring slot regions. In this regard, guard times between subsequent slot regions are introduced to compensate synchronization inaccuracies, which have to be bounded by a maximal offset $d_{\text{maxOffset}}$, and other transceiver-related factors. Transceiver-related factors include, for instance, switching delays of the

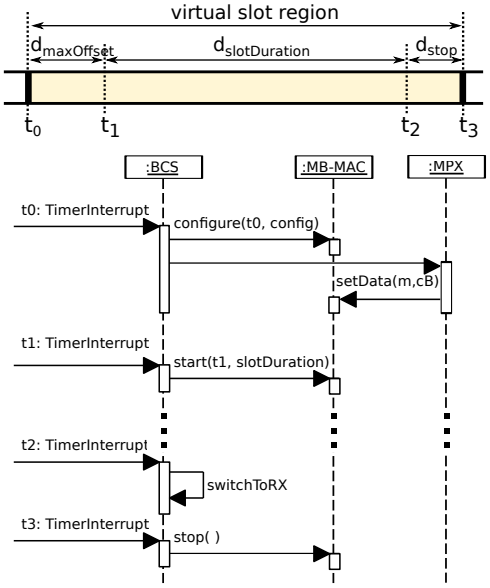

Figure 3: Structure of virtual slot regions and activation of the MAC protocol by the BCS.

CC 2420 transceiver to guarantee that every node with an assigned RX TO is in reception mode at the start of a virtual slot region. In this context $d_{TX\rightarrow RX}^{CC2420}$ denotes the maximal switching delay from transmission (TX) to reception (RX) mode, as specified by the data sheet.

Figure 3 shows the layout of a virtual slot region and the interface used by the BCS to control the execution of a MAC protocol. The duration of the region $d_{\text{slotDuration}}$ depends on the specific virtual slot region as configured in the macro slot configuration and may allow several transmissions within one region. At $t0$, the slot region starts (according to the node's local clock). At this point, the BCS first calls the `configure` function of the protocol and then executes the multiplexer (MPX), which checks for frames to be transmitted. Since the synchronization offset is bounded by $d_{\text{maxOffset}}$, all nodes have reached the start of the slot region not later than at $t1$, when the control over the medium is passed on to the MAC protocol, which now may start transmissions. At $t2$, the BCS switches the transceivers back to reception mode, to guarantee that all nodes perceive the transmissions within the bounds of the assigned slot region. Thus, all MAC protocols have to ensure that their transmissions are finished at this point in time. Therefore, the guard time d_{stop} between $t2$ and $t3$ (the locally perceived end of the slot region), is defined as $d_{\text{stop}} = \max(d_{\text{maxOffset}}, d_{TX\rightarrow RX}^{CC2420})$. By considering the maximal synchronization offset $d_{\text{maxOffset}}$ and the transceiver's switching delay $d_{TX\rightarrow RX}^{CC2420}$ from TX to RX mode, this guarantees not only that every node re-

ceives the end of a transmission within the slot region's bounds, but also that a node sending in the current slot region is ready to receive new frames at the start of the subsequent slot region.

Using a beacon-based synchronization protocol (limited to single-hop networks) instead of BBS in conjunction with a macro slot duration of $1\,s$, an upper bound for the synchronization offset can be derived with $d_{\text{maxOffset}} = 100\,\mu s$. Together with the maximal switching delay $d_{TX\rightarrow RX}^{CC2420} = 192\,\mu s$, the resulting guard time is $d_{\text{stop}} = 192\,\mu s$ and the minimal virtual slot region duration is about $5.5\,ms$ (for the MB protocol). If BBS is used to synchronize a three hop network, the maximal offset is bounded by $d_{\text{maxOffset}} = 338\,\mu s$. Hence, the guard time is $d_{\text{stop}} = 338\,\mu s$ and the minimal length of the virtual slot region is exemplarily $6.8\,ms$ for the MB protocol.

4.2 The BiPS Application Scheduler

The BAS is responsible for the execution of less time-stringent applications and higher-layer protocols and is classified as cooperative, event-based, and non-preemptive. It is small-scale, since it does not manage processes or threads but handles function callbacks to execute components. To be executed by the BAS, components have to register for relevant events, such as frame receptions or timer expirations. The events are identified by unique identifiers, which are used to request the event's execution at the next scheduling decision. Since the execution of events can be requested in interrupt mode but the BAS runs in non-interrupt mode, events, which are announced in interrupt mode, can be continued after leaving this mode. This is, for instance, used in BiPS after the reception of a frame, which is indicated by a MAC protocol running in interrupt mode, but must not entirely be handled in interrupt mode due to its possibly costly processing.

Synchronization primitives between components managed by the BAS are not required, since they work cooperatively and their execution can only be interrupted by hardware interrupts. There are, however, critical sections in which a component needs to interact with low level functions, e.g., if a message is sent via UART (Universal Asynchronous Receiver Transmitter). To prevent data races, access to shared memory has to be exclusive in these cases, which is obtained by temporarily disabling interrupts, which, in turn, can have a negative impact on time-critical protocols. To mitigate this effect, we measured the worst-case processing duration $d_{\text{intr}}^{\max}$ of all interrupt routines of the device drivers. Then, we let BCS take over control $d_{\text{intr}}^{\max}$ before a time-critical event is about to happen, which disables all non-relevant interrupts, thereby deferring the execution of less prior components. In order to keep $d_{\text{intr}}^{\max}$ low, we demand the interrupt routines to consume the shortest duration possible, which could be achieved by e.g. using direct memory access (DMA) to transfer data between main memory and devices. To further avoid undesired delays, the Imote 2's CPU provides nesting interrupts on two levels, so lower prioritized hardware drivers, e.g., the UART, can be interrupted by higher prioritized events, such as the expiration of a timer.

Though the BAS is sufficient in most instances, the absence of tasks and preemption can limit its applicability. Thus, it is also possible to remove BAS completely and replace it by another operating system which can in general be ported to the Imote 2 platform. This enables the development of multi-threaded applications and higher-layer protocols without being limited by the event system of BAS and cooperative scheduling. In detail, we integrated RIOT and FreeRTOS, both classical real-time OS with support for tasks, task priorities, synchronization primitives, and task preemption, into BiPS. The port of an integrated OS can be seen as a virtualization, because it has only limited access to the hardware and its execution is controlled. Since interrupts destined for the inner OS are assigned a lower priority than those used by the BCS, not only user level tasks can be interrupted by the BCS but also the kernel functionality of the inner OS. Therefore, timeliness and prioritization of MAC protocols over applications and higher-layer protocols is still guaranteed. The tasks running inside the OS can interact with the communication subsystem of BiPS using the multiplexer, as explained in section 3.3. The effort of porting an OS into BiPS can be compared to porting it to bare metal, except that drivers for peripheral devices are already provided by BiPS and only need an adaption layer in the port.

4.3 Memory Management and Fault Handling

The core hardware of the Imote 2 platform offers many features that can be used to speed up execution and to facilitate debugging of applications. In this regard, the availability of instruction and data caches and a Memory Management Unit (MMU) is fundamental. In BiPS, we use the MMU for detecting and analyzing common faults and provide tools to examine error situations. In particular, we can detect overwriting of code and read-only data, accidental writes to system memory such as page tables, accessing NULL pointers (by data access or jumping), and execution of invalid code (provoked by illegal jumps).

If a fault is detected, the in-system fault handler of BiPS gives information about possible reasons. It also prints a backtrace and offers a shell to examine memory content and to print the current virtual memory layout, physical memory allocations, and information about heap and stacks. There is also a GDB server included, so the GNU debugger can be invoked on a PC attached to the Imote 2 via serial line. This can give even more insight, since GDB can enrich pointer addresses by debug information.

By enabling the MMU, we can also enable the data cache of the Imote 2, which significantly speeds up the average execution time. On the downside, caches introduce jitter, which is conflicting with time-critical protocols and real-time systems. However, in BiPS, this jitter is greatly reduced for time-critical functions by using cache locking, a feature preventing individual data and instructions from being evicted from the caches.

Using the MMU also allows for shorter development times, as BiPS and its applications work on a common virtual memory layout and are therefore independent of their physical memory location. As result, an image with BiPS can be stored on non-volatile memory, but it can also be executed from SDRAM, which is much faster and saves write cycles of the flash memory.

5 EVALUATION

In this section, experiments are presented comparing BiPS with state-of-the-practice implementations for WSNs. In this regard, the focus is not on the individual protocols in BiPS, since deterministic protocols are predictable by design and CSMA/CA protocols are well studied (see, e.g., (Ziouva and Antonakopoulos, 2002; Wang and Kar, 2005; Koubaa et al., 2006)). Instead, the objective is to evaluate BiPS as framework for real-time-capable protocols. In this regard, resource usage (Sect. 5.1) and predictability of timer delays (Sect. 5.2) and delays to detect Start-Frame-Delimiters (SFDs, Sect. 5.3) are of interest.

Since there are only few OSs supporting the X-Scale PXA271 controller of the Imote 2, quantitative comparisons are severely restricted. In particular, we only found direct support for this microcontroller in TinyOS, which we used in version 2.1.2. To compare BiPS with a classical real-time OS, we added a port for the Imote 2 to RIOT (Hahm et al., 2013a). But since RIOT does not provide a driver for the CC2420 transceiver, comparisons are restricted to communication-independent aspects. In all experiments, the Imote 2 runs with 104 MHz.

5.1 Memory Usage

In general, memory usage of a program can be divided into static and dynamic memory. Static memory is the amount of memory that is needed by the application image and consists of code plus read-only and pre-initialized read-write data. It can be measured by taking the file size of the image after linking all object files. Dynamic memory is the memory needed while executing the program. It covers all read-write data that can be seen by static analyzes of the program, such as global variables, but there are also data, whose size cannot be determined in advance, such as heap and stack. It is, therefore, not trivial to give meaningful numbers for dynamic memory, but one can provide minimal requirements.

The Imote 2, compared to other sensor platforms, is very powerful: It offers 32 MiB flash memory, which is an upper limit for static memory usage, as well as 256 KiB SRAM and 32 MiB SDRAM, which are upper limits for dynamic memory usage. Because of this large amount, optimizing our applications for size was a minor objective. The features we employed, such as virtual memory handling and a comprehensive fault handler (see Sect. 4.3), consume about 59 KiB static memory and 26 KiB dynamic memory, of which 16 KiB are contributed to the MMU table alone.

As minimum requirement, a BiPS application needs 140 KiB static and 26 KiB dynamic memory. An application using BiPS excessively requires about 350 KiB static and 35 KiB dynamic memory. Note, however, that by compiling for size instead of performance, the static size of applications can already be reduced by 20%. If size should matter (remember: even with a large application, we only use 1% of static and 0.1% of dynamic memory), we could also drop features or compile the application using an alternative size-optimized instruction set offered by the processor.

5.2 Timer Delays

In real-time systems, delays to react to external events must be low and predictable. At best, these delays are bounded and load-independent. In the following, we assess delays to process timers, whose expiration is a special type of event. They are used representative for other types of events, e.g., caused peripheral devices.

The experiment's setup consists of one Imote 2, on which TinyOS, RIOT, and BiPS are installed consecutively. In each case, there is support for (at least) two timer variants, which are evaluated independently: One variant, where the expiration is processed in in-

Table 1: Timer delays.

		w/o load						with load					
		time [µs]			instructions			time [µs]			instructions		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max
TinyOS	Timer	18.1	18.3	18.6	177	177	177	19.3	46.2	1316	192	497.2	15382
	Alarm	13.5	13.6	13.7	112	112	112	13.5	16.8	18.4	112	112	112
RIOT	virtual timer	22	22	22	663	663	663	26.7	49.9	63.4	933	933.1	1254
	hw timer	2.3	2.4	2.4	49	49	50	2.3	3.5	7.6	49	49.3	50
BiPS	event timer	2.3	3.3	7.9	49	49.3	50	5.3	8.8	594.2	53	86.9	13260
	hw timer	1.5	1.6	1.6	13	13	13	1.5	1.7	2.5	13	13.3	14

terrupt context – called `Alarm` (TinyOS) or hw timer (RIOT, BiPS) –, and one variant with execution in non-interrupt mode – called `Timer` (TinyOS), virtual timer (RIOT), or event timer (BiPS). The time to process a timer is measured by performance counters of the Imote 2. For each OS and timer variant, experiments are done without and with background load and repeated 10,000 times.

The results are shown in Table 1. Without load, there is no concurrency. The results show that the processing of timers in interrupt mode is, in general, faster and less variable than in non-interrupt mode. Due to the costly runtime model and its distinctive abstraction layers, hardware timer delays with TinyOS are in general larger than with RIOT and BiPS. The lowest and most stable delays are achieved by hardware timers of RIOT and BiPS, since they benefit from optimized implementations. Perhaps most surprising are the virtual timers of RIOT, whose execution implies the preemption of an idle task and a context switch, which sums up to $22\mu s$ on average.

When additional random background load is generated by a second timer[3], evictions of data from the Imote 2's instruction and data caches are provoked. Corresponding results show a deterioration of delays; in particular, for timers processed in non-interrupt mode. In this regard, processing delays in RIOT increase the least due to the priority-based and preemptive scheduling strategy. On the other hand, non-interrupt timers of TinyOS and BiPS suffer from the cooperative scheduling strategies, yet BiPS outperforms TinyOS due to better exploitation of hardware features like data caches. In contrast to timers processed in non-interrupt mode, the increase of delays with hardware timers is much lower. Here, the smallest and less variable times are achieved with BiPS due to the usage of cache locking. Thus, reactions to timer events are most predictable with BiPS and hardware timers, which are therefore the best choice for the execution of protocols with time-critical behavior.

[3]This timer is processed in non-interrupt mode and with lowest priority (if priorities are supported by the OS).

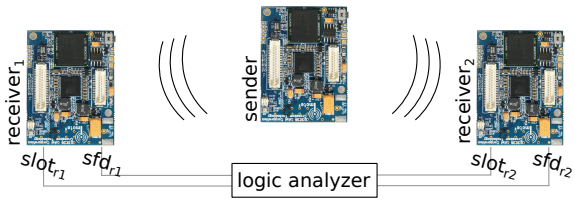

Figure 4: Experiment setup with three Imote 2 (one sender and two receivers) to evaluate the accuracy of SFD-based synchronization and medium slotting.

5.3 SFD-based Synchronization

According to the CC 2420's datasheet, the processing delay at receivers when detecting an SFD is typically 3 µs (TI, 2013). However, this time does not include the delay until an SFD is recognized in software, which additionally worsens the accuracy if execution delays differ. To detect an SFD with low latency, the CC 2420 provides an output pin signalizing the detection of an SFD. This pin is connected to a GPIO (General Purpose Input/Output) pin of the microcontroller, thereby enabling the triggering of hardware interrupts.

In the following, the time difference in detecting an SFD between several nodes is evaluated, where the comparison is restricted to BiPS and Tiny OS, since RIOT does not provide a CC 2420 driver. By comparing delays at two receivers, we ensure that the same piece of code is executed on all nodes of interest and all results can be ascribed to different detection delays in the nodes' transceiver and variations in execution times.

The conducted experiment consists of three Imote 2 (see Fig. 4): One sender and two receivers. The receivers are connected via two GPIO pins with a Sigma2 logic analyzer[4] running with a sampling rate of 100 MHz. After detecting a rising edge at the SFD pin of the transceiver, the receivers raise the GPIO pin sfd$_{r1}$ and sfd$_{r2}$, respectively.[5] The second GPIO pin

[4]http://www.asix.net/dbg_sigma.htm

[5]In Tiny OS, we extended the CC2420ReceiveP and CC2420TransmitP component for this purpose.

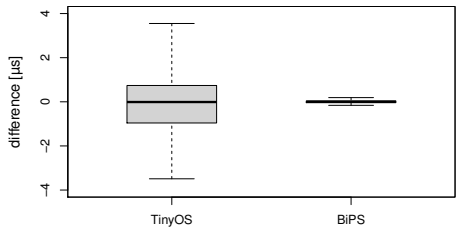

Figure 5: Accuracy of SFD detection between both receivers.

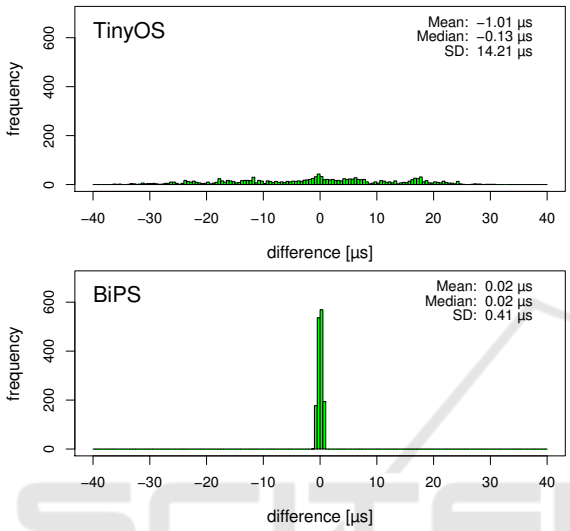

Figure 6: Accuracy of the detection of slot starts between both receivers.

($slot_{r1}$, $slot_{r2}$) is raised 10 ms after the detection of the SFD to emulate virtual medium slotting and to evaluate the accuracy of slot bounds. For this purpose, a timer is activated whose expiration time is based on a timestamp, which was stored when detecting the SFD.[6] Besides the control of GPIO pins, no further application is running. In total, 1500 observations are made with BiPS as well as TinyOS.

The results are shown in Figs. 5 and 6. In Fig. 5, the accuracy of detecting SFD events in software is presented. Here, the median of difference is for BiPS as well as TinyOS almost 0μs; as expected due to the usage of identical receivers. However, the variation with TinyOS is much larger than with BiPS (standard deviations 1.5μs vs. 0.06μs), thereby indicating that the detection accuracy of SFDs with TinyOS suffers from diverging execution delays. With BiPS, on the other hand, delays are almost constant.

Figure 6 presents the differences in the perception of slot starts 10 ms after the SFD event. Here, the differences with TinyOS are very large and spread within a range of [−36.23μs, 30.58μs], where differences with BiPS are almost constant. Besides the more inaccurate detection of SFDs, the significant deterioration with TinyOS is mainly caused by the missing usage of the Imote 2's μs-hardware timers, which are fully exploited by BiPS but not by TinyOS.

[6]Timestamp is taken early at driver level.

5.4 Portability

Since BiPS is a tailored OS for the Imote 2 platform and highly optimized for the PXA271 processor, it is compartively hard to port it to another platform. More precisely, the effort depends on the features, which the target platform can offer. If e.g., the platform does not provide an MMU, the whole protection and debugging facility of BiPS cannot be ported. On the other hand, if an MMU is provided, a deep knowledge of the core hardware is needed to port the memory management of BiPS to the target.

Also, if caches are provided but cache locking is missing, the jitter of time-critical events will worsen due to cache eviction. On the other hand, if there are no caches at all, jitter will reduce, but the overall execution time will rise. Thus, the results obtained in Table 1 cannot easily be assumed for all platforms.

However, since the modularity of the architecture, as seen in Fig. 1, is retained in the source code structure, it is possible to omit or replace the parts, which are needed to adapt the framework to a new platform.

6 RELATED WORK

In the literature, several evaluations can be found (e.g., in (Harvan and Schönwälder, 2008; Silva et al., 2009; Basmer et al., 2011)) showing that the behavior and performance of a protocol highly-depends on the implementation. In this regard, the used OS, which is in the WSN domain often TinyOS or Contiki (Dunkels et al., 2004), is crucial. However, most protocol implementations do not scrutinize the handling of concurrency in the used OS. Thus, it is often unclear – in particular, if a protocol is announced to support real-time guarantees – how timing constraints of the protocol are satisfied and if provided prioritization measures are actually sufficient. In the following, our outline of related work first covers common OSs used for protocol implementations and compares their suitability for real-time applications. Afterward, we relate the capabilities of BiPS as protocol framework with various MAC protocols with existing solutions.

The most common OS for WSNs is probably TinyOS, an extensive and popular OS ported to many hardware platforms. TinyOS is event-driven and tasks are scheduled cooperatively in FIFO or EDF order.

Tasks are not preempted by other tasks, although interrupt handlers, commands, and events can preempt a task. Since version 2, there is also support for full multi-threading and synchronization primitives, yet w.r.t. the Imote 2 platform, some features like MMU and data caches are no longer used. TinyOS does not offer a mechanism to run the communication stack in a privileged context, such that protocols with stringent timing constraints are not easy to implement. Additionally, the modular structure of TinyOS increases the amount and variance of execution delays (see Sect. 5.2). Applications for TinyOS are written in a specialized C dialect, called nesC, rendering the learning curve steeper for C/C++ programmers.

Another OS, RIOT, was designed for the Internet of Things, with a special focus on real-time communication (Hahm et al., 2013b). RIOT comes with full multi-threading support and uses IPC (Inter-Process Communication) to pass messages between modules, but does not offer memory protection or advanced measures on fault detection. The implementation of time-critical protocols still demands some programming tricks. Though there is no native port of RIOT for the Imote 2, it can run on this platform as extension of BiPS, where all RIOT processes are scheduled by the BAS of BiPS. Like BiPS, RIOT offers support for C(++).

Another well-known OS is Contiki (Dunkels et al., 2004), focused on heavily resource-constrained devices. Processes are scheduled cooperatively, although there is a mechanism for light-weight multi-threading, called Protothreads, and also an additional library offering full multi-threading support. Real-time tasks can be employed for time-critical code and a lot of additional functionalities are given. However, memory protection has never been a goal of Contiki. Programs for Contiki are written in constrained C, i.e., not the full language is supported, with a heavy use of C macros.

W.r.t. BiPS's contribution as protocol framework, there is up to our knowledge no approach with a similar flexible integration of multiple MAC protocols. Existing approaches are instead severely restricted in the number and activation of supported protocols. This holds both for established communication standards like IEEE 802.15.4 (IEEE, 2011), WirelessHART (IEC, 2010), and ISA 100.11a (IEC, 2012), and for proposals from research like Z-MAC (Rhee et al., 2008), ER-MAC (Sitanayah et al., 2010), and (Gilani et al., 2013). In general, all proposals establish some kind of time frame (often called superframe) and combine TDMA- with CSMA/CA. The pattern of allocated contention-based/-free regions is often fixed or time slots have constant length. Thus, compared to virtual slot regions in BiPS, the individual protocols' application is highly constrained. Furthermore, they are limited to one reservation-based and one best effort protocol, and do not support further classes of protocols.

7 CONCLUSION

This paper presents BiPS, a real-time-capable protocol framework for WSNs based on the Imote 2 platform. The key contributions of BiPS are twofold: First, from a protocol design perspective, it provides various MAC protocols, interfaces for the flexible usage of them, and an infrastructure to add further MAC protocols. Thereby, BiPS enables the usage of different MAC protocols for different message types in one scenario. This helps to fit the communication requirements and characteristics in an optimal way with minimal overhead (e.g., reserved bandwidth). Second, from an implementation perspective, it brings support for OS functionalities to run real-time-capable protocols and less time-critical applications in compliance with time constraints of the protocols. In a series of experiments, this paper shows that with BiPS, the predictability of delays to process external events is significantly improved. Thereby, evidence is provided that BiPS outperforms existing solutions w.r.t. the suitability for real-time systems.

Different from state-of-the-practice protocol implementations, our approach is not based on a universal (real-time) OS but on a specially developed light-weight and protocol-centric OS. We justify this decision by the experience that compliance with time constraints is one of the most crucial tasks of implementations in distributed real-time systems and that the support of multiple processes/tasks with extensive synchronization measures is less important and is actually often not needed in typical WSN applications. However, we also admit that this makes BiPS a very specific solution, dedicated to applications with strict real-time requirements regarding communication and reaction time. This also holds for the required tailoring for the used hardware platform needed to meet the real-time constraints, which on the other hand implies drawbacks and additional efforts regarding the portability.

In our future work, we are going to exploit the usage of multiple channels. Furthermore, we will shift the focus to higher layers (e.g., a QoS routing protocols and service-based middleware) for which BiPS will serve as fundament.

REFERENCES

Basmer, T., Schomann, H., and Peter, S. (2011). Implementation Analysis of the IEEE 802.15.4 MAC for Wireless Sensor Networks. In *Mobile and Wireless Networking (iCOST), Int. Conference on Selected Topics in*.

Braun, T., Christmann, D., Gotzhein, R., and Mater, A. (2014a). SDL Implementations for Wireless Sensor Networks - Incorporation of PragmaDev's RTDS into the Deterministic Protocol Stack BiPS. In *System Analysis and Modeling: Models and Reusability - 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings*, volume 8769 of *Lecture Notes in Computer Science*, pages 271–286. Springer.

Braun, T., Gotzhein, R., and Kuhn, T. (2014b). Mode-based Scheduling with Fast Mode-Signaling – A Method for Efficient Usage of Network Time Slots. *Journal of Advances in Computer Networks (JACN)*, 2:48–57.

Christmann, D., Gotzhein, R., and Rohr, S. (2012). The Arbitrating Value Transfer Protocol (AVTP) - Deterministic Binary Countdown in Wireless Multi-Hop Networks. In *Computer Communications and Networks (ICCCN), 21st International Conference on*.

Cunha, A., Koubaa, A., Severino, R., and Alves, M. (2007). Open-ZB: an Open-source Implementation of the IEEE 802.15.4/ZigBee Protocol Stack on TinyOS. In *IEEE 4th Int. Conference on Mobile Adhoc and Sensor Systems, MASS 2007, 8-11 October 2007, Pisa, Italy*, pages 1–12. IEEE Computer Society.

Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE Int. Conference on*, pages 455–462.

Elyengui, S., Bouhouchi, R., and Ezzedine, T. (2015). LOADng Routing Protocol Evaluation for Bidirectional Data flow in AMI Mesh Networks. *CoRR*, abs/1506.06357.

Farooq, M. O. and Kunz, T. (2011). Operating Systems for Wireless Sensor Networks: A Survey. *Sensors*, 11(6):5900–5930.

Gilani, M. H. S., Sarrafi, I., and Abbaspour, M. (2013). An Adaptive CSMA/TDMA Hybrid MAC for Energy and Throughput Improvement of Wireless Sensor Networks. *Ad Hoc Networks*, 11(4):1297–1304.

Glaropoulos, I., Vukadinovic, V., and Mangold, S. (2014). Contiki80211: An IEEE 802.11 Radio Link Layer for the Contiki OS. In *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, pages 621–624.

Gotzhein, R. and Kuhn, T. (2011). Black Burst Synchronization (BBS) – A Protocol for Deterministic Tick and Time Synchronization in Wireless Networks. *Computer Networks*, 55(13):3015–3031.

Hahm, O., Baccelli, E., Günes, M., Wählisch, M., and Schmidt, T. C. (2013a). RIOT OS: Towards an OS for the Internet of Things. In *INFOCOM*.

Hahm, O., Baccelli, E., Günes, M., Wählisch, M., and Schmidt, T. C. (2013b). RIOT OS: Towards an OS for the Internet of Things. In *32nd IEEE Int. Conference on Computer Communications (INFOCOM), Poster Session*.

Harvan, M. and Schönwälder, J. (2008). TinyOS Motes on the Internet: IPv6 over 802.15.4 (6lowpan). *Praxis der Informationsverarbeitung und Kommunikation*, 31(4):244–251.

Hauer, J.-H. (2009). TKN15.4: An IEEE 802.15.4 MAC Implementation for TinyOS 2. Technical Report TKN-08-003, Technical University Berlin.

IEC (2010). Industrial Communication Networks - Wireless Communication Network and Communication Profiles - WirelessHART (IEC 62591 ed 1.0).

IEC (2012). Industrial Communication Networks - Wireless Communication Network and Communication Profiles - ISA 100.11a (IEC 62734 ed 1.0).

IEEE (2011). *IEEE Standard 802 Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Computer Society, New York, NY, USA.

ISO (2004). Controller Area Network (CAN), ISO 11898.

ITU-T (2012). ITU-T Recommendation Z.100 (12/11) - Specification and Description Language - Overview of SDL-2010.

Kopetz, H. (1997). *Real-Time Systems – Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers.

Koubaa, A., Alves, M., and Tovar, E. (2006). A Comprehensive Simulation Study of Slotted CSMA/CA for IEEE 802.15.4 Wireless Sensor Networks. In *Factory Communication Systems, IEEE Int. Workshop on*.

MEMSIC Inc. (2013). Imote2 datasheet.

Pereira, N., Andersson, B., Tovar, E., and Rowe, A. (2007). Static-Priority Scheduling over Wireless Networks with Multiple Broadcast Domains. In *RTSS '07: Proceedings of the 28th IEEE International Real-Time Systems Symposium*, pages 447–456, Washington, DC, USA. IEEE Computer Society.

Rhee, I., Warrier, A., Aia, M., Min, J., and Sichitiu, M. L. (2008). Z-MAC: a Hybrid MAC for Wireless Sensor Networks. *IEEE/ACM Trans. Netw.*, 16(3):511–524.

Silva, R., Silva, J. S., and Boavida, F. (2009). Evaluating 6LowPAN Implementations in WSNs. *Proceedings of 9th Conferncia sobre Redes de Computadores Oeiras, Portugal*, 21.

Sitanayah, L., Sreenan, C. J., and Brown, K. N. (2010). Emergency Response MAC Protocol (ER-MAC) for Wireless Sensor Networks. In Abdelzaher, T. F., Voigt, T., and Wolisz, A., editors, *Proceedings of the 9th Int. Conference on Information Processing in Sensor Networks, IPSN, Stockholm, Sweden*. ACM.

TI (2013). CC2420 Datasheet. Revision SWRS041c.

Wang, X. and Kar, K. (2005). Throughput Modelling and Fairness Issues in CSMA/CA-based Ad-hoc Networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 23–34.

Ziouva, E. and Antonakopoulos, T. (2002). CSMA/CA Performance under High Traffic Conditions: Throughput and Delay Analysis. *Computer Communications*, 25(3):313 – 321.