EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

CLAWS: Cross-Layer Adaptable Wireless System enabling full cross-layer experimentation on real-time software-defined 802.15.4

Bertold Van den Bergh[*][†], Tom Vermeulen[†], Marian Verhelst and Sofie Pollin

Abstract

Motivation: In recent years, researchers have developed a large and growing set of protocols and algorithms to improve the throughput and capacity of wireless networks. These schemes span the physical (PHY), medium access control (MAC), and higher layers of the protocol stack. Most effective innovations however require cross-layer modifications of both PHY and higher layers. To date, the verification of those designs has been limited to simulations or small setups relying often on off-line processing of the results. MAC layer results that rely on even the tiniest modification of the PHY can only be verified under simplified networking assumptions. Similarly, novel PHY algorithms are typically only verified for a single wireless link, avoiding complex scenarios. Most importantly, there is almost no cooperation between PHY and networking communities, as the tools and testbeds they use are incompatible.

Contributions: In this paper, we propose a methodology for fully flexible PHY, MAC, and network layer verification that is designed to (a) reuse existing software components from PHY and network communities, (b) enable both simple- and expert-level modification and configuration of all components, (c) have real-time performance benchmarked with off-the-shelf systems, and (d) enable large networking experiments including off-the-shelf nodes for rapid experimentation, testing, and comparison. The main contribution of this paper is the introduction of an approach that enables the realization of full software-defined radio (SDR) sensor nodes, all running on a single field-programmable gate array and reusing PHY layer SDR tools and typical operating systems such as Contiki OS. Subsequently, the paper will illustrate the strengths of the proposed approach by demonstrating communication with off-the-shelf sensor nodes. This allows fair benchmarking with state-of-the-art or off-the-shelf solutions. Finally, some cross-layer improvements are proposed and compared with the baseline off-the-shelf system. This proves our claims that the proposed platform is a very useful tool for cross-layer experimentation, in that it allows full cross-layer control of the PHY and network layers, and moreover enables elegant comparison with state-of-the-art designs. This architecture is provided to the open source community (http://claws.be/), in order to become a framework for validating and benchmarking wireless cross-layer innovations.

Keywords: Software-defined radio; Performance evaluation; Cross-layer design

*Correspondence: bertold.vandenbergh@esat.kuleuven.be
†Equal contributors
KU Leuven, ESAT-TELEMIC, Kasteelpark Arenberg 10, Heverlee, Belgium

1 Introduction

The flexibility and ubiquity of wireless communication solutions played a very significant role in the tremendous growth of mobile devices such as smartphones and tablet PCs and as such has been an important driver for technology breakthroughs in the last decade. The next, fifth generation of communication solutions (5G) and the Internet-of-Things will require a radical rethinking of the wireless communication landscape to keep improving the spectral and energy efficiency at acceptable cost. State-of-the-art communication solutions already operate close to Shannon capacity. Therefore, in order to support ever-increasing throughput requirements, modern solutions mainly employ two strategies. The first strategy is to use more frequencies which requires increased flexibility at the radio layer, enabled by software radio. The other strategy is to use a more distributed approach; this means that networks are built using many very small cells that cooperate. This, naturally, requires complex networking protocols. Future 5G communication innovations will hence revolve around joint innovation across all layers of the protocol stack, strongly requiring a robust approach for performance evaluation of such complex cross-layer designs. By their very nature, cross-layer communication solutions require mixing different disciplines. As a result, modeling, design, and testing should jointly consider analog, digital, baseband, RF, hardware, and software, resulting in high system complexity even at the level of a single radio. Comparing cross-layer solutions is often achieved by comparing algorithm A on hardware X with algorithm B on hardware Y in non-real-time ideal lab settings. Consequently, performance comparisons are confusing and only hold as far as the calibration methods and assumptions hold. In addition to that, implementation of full communication systems requires a very broad expertise, and it becomes impossible for a single researcher to know sufficiently well all aspects relevant for cross-layer design and implementation.

In this paper, a Cross-Layer Adaptable Wireless System (CLAWS) is proposed that enables gradual improvement and evaluation of cross-layer design innovations. CLAWS is a fully flexible communication node and constructed by combining research tools from both the physical (PHY) and networking community. CLAWS is designed to be user-friendly. Simple experiments can be performed with little knowledge, while experts can still access and modify the core functionality. At the PHY level for example, it is possible to (1) tune parameters of existing functional blocks or (2) add novel functional blocks which requires more experience with the PHY layer (field-programmable gate array (FPGA)) tools. A similar approach can be followed when considering the medium access control (MAC) and network layers, which allow for improved protocol implementations on a default PHY or alternatively take advantage of the extended PHY functionality. By doing so, the radio can emulate (1) an off-the-shelf radio when standard functional blocks and parameter settings are chosen or (2) a cross-layer improved radio when parameters of existing functional blocks are tuned or novel functionality is added. In this paper, a relevant cross-layer design will be introduced that relies on a novel PHY block for digital mixing or frequency shifting, which is then exploited in user space by a multi-channel MAC. The novel protocol is easily realized in CLAWS, with minimal development effort, and elegantly compared with the single-channel state of the art using the same hardware and network context. This enables effective and correct performance evaluation of the functional improvement, independent from hardware or context calibration errors.

In summary, the proposed methodology in this paper promises to facilitate and benchmark cross-layer radio designs in various large-scale distributed setups. Below, we first detail the state of the art with respect to experimental performance evaluation of wireless solutions. Then, in Section 3, the proposed sensor node architecture is introduced. Section 4 discusses the performance of our design, and Section 5 finally presents a small yet relevant cross-layer improvement that could be implemented extremely fast and benchmarked elegantly using our setup.

2 State of the art for user-friendly cross-layer experimentation

To enable realistic and repeatable verification of cross-layer innovations, spanning PHY, MAC, and network layers, it becomes necessary to test the cross-layer improved setup and compare it with an off-the-shelf setup in a similar context. In addition, the methodology should reuse and combine as much as possible research and prototyping tools from PHY and networking communities. Below, we summarize the state of the art with respect to software-defined-radio (SDR) experimentation and experimentation using off-the-shelf radios. Having made this comparison, we illustrate how our methodology wants to improve on that.

2.1 Software-defined radio approaches

At the core, our methodology relies on SDR [1]. Typical software-defined radios can operate in almost any frequency bands using almost any wireless communication standard. They use a combination of FPGAs, digital signal processors (DSPs), and versatile analog/RF designs to achieve this level of system performance across a range of radio standards. The SDR's core functionality can be changed by modifying the software and firmware on top of the hardware. Various research groups are embracing the availability of off-the-shelf SDR solutions as a means of showing the ideas and algorithms at work [2]. Many recent innovative ideas have been proven or introduced by means of SDR solutions, e.g., full duplex [3]. These experiments are however, typically, limited to small setups and involve off-line processing of the results, as state-of-the-art SDR approaches do not allow meeting stringent delay requirements and/or often do not implement the full physical layer. To the best of our knowledge, we are not aware of any SDR implementation that allows networking with off-the-shelf radios (allowing to scale up the size of the experiments to hundreds or thousands of nodes at reasonable cost) which requires (a) a real-time PHY implementation and (b) compatibility with common protocol stacks.

Real-time SDR operation is challenging because of processing and data communication delays. In its simplest form, a SDR can be used as an analog front end to convert the radio signal to digital samples. A FGPA then processes the samples further. Most SDRs have a small FPGA that only implements digital filtering and down-conversion. These digital samples are sent over a connection (USB, Ethernet, PCIe, etc.) to a host PC to be processed, which causes long delays (communication delays and processing delays). The measured latencies range from 1 ms up to 30 ms [4]. Obviously, these high latencies limit the response time and precise timing control needed in a MAC design (e.g., the default acknowledgment (ACK) timeout is 48 μs in IEEE 802.11b and 864 μs in the 2.4 GHz IEEE 802.15.4). As such, the communication latency between SDR and host prohibits the development of time-critical MAC solutions.

Nychis et al. [5] present a split functionality for streaming SDR platforms. They have concluded that time-critical radio or MAC functions should not be placed on the host but as close to the radio as possible. They annotate the sample stream with timestamps and control information, allowing them to avoid some, but not all, latency problems. Their design only allows slotted MAC implementations for streaming implementations, as turnaround times cannot be optimized with their approach, making more dynamic networking conditions where nodes contend in real-time impossible. A similar approach is taken by Puschmann et al. [6], where a MAC framework is built on top of the IRIS SDR testbed. While interesting, these approaches only allow experimental verification of some networking scenarios, limited to a small number of expensive SDR nodes and often not in real-time. This limits the possibilities for exploring most non-trivial higher layer protocols.

Bloessl et al. [7] created a GNU Radio-based [8] IEEE 802.15.4 implementation using the USRP N210 hardware. They implement the PHY, MAC, and network layer in GNU Radio which runs on the host PC. This setup requires relatively cheap RF hardware but offloads all computation to a host computer via a gigabit Ethernet connection. A very powerful computer is required to fully exploit the flexibility. The USRP N210 is able to stream 25 MHz of I and Q samples in both directions, but processing this amount of data is extremely computationally intensive. Moreover, the implementation is not complete as they did not implement carrier sensing and the carrier sense multiple access with collision avoidance (CSMA/CA) protocol.

To address these issues, parts or even all of the processing can be done by the FPGA on the SDR platform. This approach completely mitigates the latency and processing issues but introduces other shortcomings as programming these devices requires a very deep understanding of the underlying hardware and they require domain-specific knowledge (HDL programming, signal processing expertise). As protocol experts often lack knowledge of hardware programming languages or baseband functionalities, it is hard for them to use these platforms or even change the physical layer. While SDR approaches that expose MAC functionality exist, such as WARP, the developed drivers and MAC protocols are platform dependent [9]. Furthermore, these drivers and protocols are provided by the SDR or physical layer community and, since they are often custom non-standard protocols, not widely known to the networking research community. These designs are often considered too limited in functionality for protocol researchers, which are used to working with full protocol stacks, e.g., interoperability with a whole range of off-the-shelf radios is expected, and extended protocol tuning options are desired. Ideally, the SDR should be compatible with existing protocol stacks developed in other communities, enabling to combine innovative PHY designs from the PHY-SDR community with protocol developments from the networking community. This avoids duplicating development work and allows each community to leverage upon its own tools and strengths.

The above constraints limit the use of these SDR platforms towards research on the PHY layer, potentially combined with very simple, not time critical, MAC layers. As such, the basic promise of SDRs (reconfigurable connectivity) is still unobtainable: (a) full open SDR protocol stacks that include MAC protocols are still missing; (b) full operation with complete networking layers (including MAC and routing) is not realizable; and (c) real-time SDR interaction allowing benchmarking with off-the-shelf nodes and the realization of large testbeds has currently not been achieved.

A similar approach as proposed in this paper was proposed in [10], which is a real-time 802.11g PHY and MAC implementation on FPGA. The design focuses on design reusability, and many of the relevant parameters were chosen to be easily configurable. By using Bluespec, user-friendly hardware programming is achieved. A CSMA/CA MAC is implemented on the FGPA, as well as a soft processor that can possibly run higher layer protocol stacks. While this design is in principle very similar to the method we propose here, they never verified that their design can work with off-the-shelf radios and never ported higher layer protocol stacks on the SDR. In this paper, we provide a full 802.15.4 PHY and MAC layer and also a hardware abstraction layer that allows porting most OS or protocol stacks on our PHY. In addition, we benchmark our design with off-the-shelf radios and realize a relevant cross-layer improvement to illustrate the key benefits of the proposed approach.

2.2 Off-the-shelf radio approaches

In addition to PHY layer testbeds, wireless communication test facilities with hundreds of radios exist, such as the ORBIT testbed [11] or the w-iLab.t testbed [12]. The wireless communication nodes consist of off-the-shelf communication solutions such as IEEE 802.11 with little or no flexibility as this is constrained by flexibility offered by the chip implementation. As a result, these testbeds can only be used for higher layer protocol or application research. Key in those approaches is the selection of the appropriate OS and protocol stack for higher layer networking research; often, the challenge is obtaining fast and fine-grained control of the various possible radio chipsets available.

In an effort to solve this challenge, generic flexible MAC approaches, originating from the SDR philosophy, have been proposed, which can be interpreted on the device itself [13,14]. Based upon the analysis of CSMA, TDMA, and hybrid MAC protocols, the decomposable MAC frameworks define a set of MAC functionalities (blocks) as a library. By combining these blocks using a wiring engine, a wide range of protocols can be realized, nevertheless limited to the predefined MAC blocks in the library. These blocks are however mostly limited by the PHY capabilities, and more powerful MAC innovation would be possible if selected PHY layer implementations would be available such as interference analysis, full duplex or flexible bandwidth, and frequency tuning.

To maximize MAC innovation capabilities, approaches exist that try to expose as much of the PHY flexibility as possible to the user space. This is not straightforward as it requires extensive knowledge of the specific OS, hardware platform, and radio controller implementation. A project that pioneered in the development of a possible solution is the FLAVIA project [15]. The project investigates how to execute MAC and (a subset of) PHY commands without the need to access the firmware of the radio device. This way, the construction of the MAC and some PHY commands can be implemented in highly reusable modules and functionalities. Furthermore, virtualization of radio resources was investigated such that several wireless network stacks could operate in parallel over the same wireless link. The approach should be extended towards SDR, where more control of the radio is possible, and virtually unlimited MAC design freedom is possible.

Considering sensor networks specifically, an OS for low-power embedded devices is often used to manage wireless devices, using protocols such as IEEE 802.15.4. TinyOS [16] and Contiki [17] are commonly used operating systems for wireless sensor nodes. Many useful and well-known higher layer protocol innovations are already implemented in these operating systems, and the development effort is mainly shifted to the radio-specific control interface. Many developers have made their own set of customizations in order to optimize the control of the radio chipset, but these are obviously not interoperable between chipsets. These customizations are often stored in private repositories and are not made publicly available. In this paper, we show that it is possible to write a generic SDR interface layer that allows running the Contiki SDR stack on top of our SDR PHY.

3 The CLAWS architecture

The CLAWS platform enables cross-layer experimentation and benchmarking at various levels of configuration complexity. In this section, we will describe the different modules of CLAWS, starting with a high-level overview of the architecture.

3.1 Overview

The architecture of the CLAWS platform is given in Figure 1. It is clearly shown how the radio front end, baseband PHY, MAC, and control functionalities are split. The baseband PHY is written in LabVIEW FPGA [18] and implemented on the FPGA to ensure real-time performance. This PHY is interfaced to a MAC processor which forms a bridge between the PHY and a Microblaze FPGA-mapped softcore processor [19]. This generic SDR interface layer provides full control over the PHY, as is needed for cross-layer innovations. In our implementation, we have ported Contiki OS to this Microblaze core in order to provide experimenters with a standard and commonly used network layer environment. Hence, we can leverage all features and innovations provided by Contiki OS. When doing pure PHY layer research, the Microblaze subsystem is not needed and can be disabled. The management application running on the host computer then listens on a TCP socket that allows direct control of the radio physical layer by sending low-level control commands through our command processor, as further explained in Section 3.5. This host control can be used for automated performance testing of the PHY and link layers when networked tests are not needed.

To build our prototype, we used the NI USRP-2942 [20] which contains a Xilinx Kintex 7 FPGA. The front end has an RF range from 400 MHz to 4.4 GHz and a sample rate

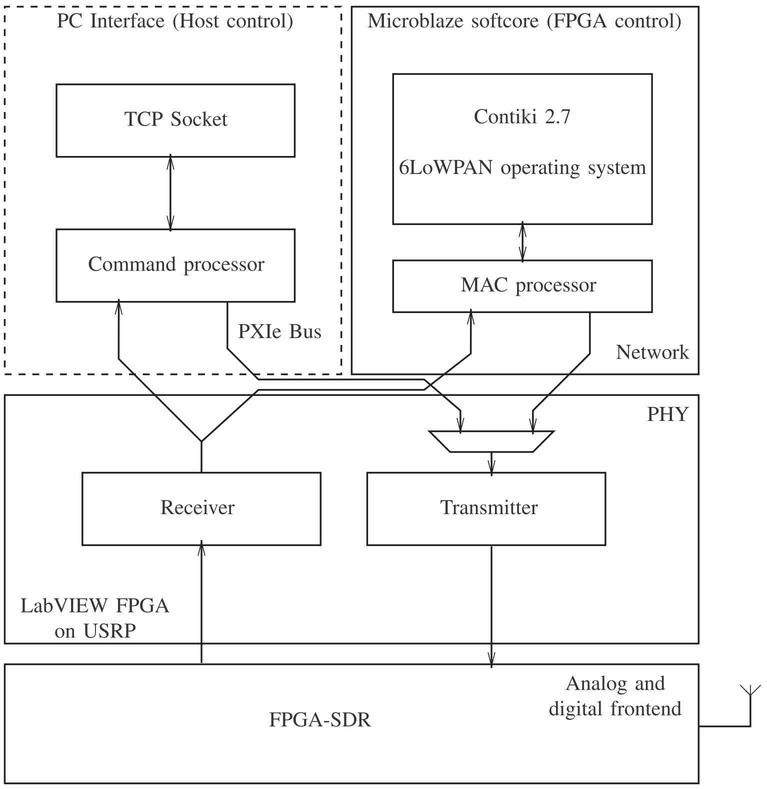

Figure 1 CLAWS architecture. CLAWS architecture showing the RF/PHY/networking functionality.

of 120 MS/s. The FPGA is connected to the host PC by a PCIe connection. Since our architecture is generic, the hardware is not fixed. It is possible to easily port the PHY layer to any other equivalent NI hardware. All functionality above the PHY, such as the Microblaze subsystem or the MAC processor, can be easily ported to other Xilinx FPGA-based SDRs as they do not depend on LabVIEW FPGA.

3.2 User-friendly FPGA-based PHY

Both the transmitter and receiver of our baseband PHY implementation are written in LabVIEW FPGA. The provided functional blocks are parameterized, and these parameters can be tuned by the host or embedded controller without recompilation. In addition, expert upgrades of the PHY are possible, to provide novel or improved functionality, but this requires updates in the LabVIEW FPGA code and recompilation of the design. We will first discuss the transmitter and then the receiver functional blocks.

3.2.1 FPGA PHY transmitter

Figure 2 shows the proposed modular PHY transmitter implementation of IEEE 802.15.4. Table 1 lists the configuration parameters that are provided to the embedded MAC protocol or to the host. To emphasize the configurability and flexibility of the design, the parameters are split in 'Standard parameters' which are available on most off-the-shelf chipsets and 'Extended parameters' which are provided in CLAWS, as we believed they were

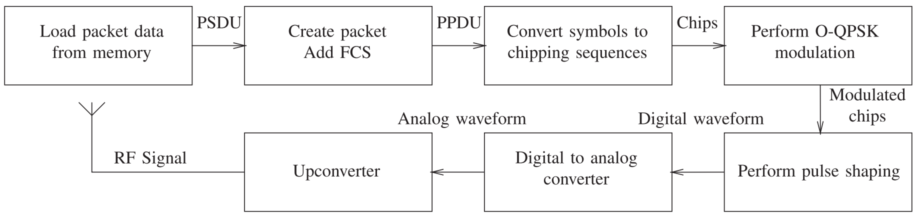

Figure 2 LabVIEW FPGA PHY TX. Block diagram of the PHY transmitter.

Table 1 FPGA PHY TX parameters

	Standard parameters	Extended parameters
Read from memory		Origin of data
Form packet	Packet length	Change CRC
	Add CRC	Change SFD
		Maximum packet length
Symbol to chip		Chipping sequence
OQPSK mod		Modulation type
Pulse shaping		Pulse shape
Sampling		Sampling rate
Upconvert	Channel number	Any frequency

These are all parameters that can be changed on the CLAWS transmitter. The first column corresponds to the blocks of Figure 2. If a cell is empty, this means nothing can be changed to this block.

relevant for most cross-layer experimentation, beyond the constraints of the IEEE 802.15.4 standard. By relying on the more capable SDR technology, we can allow a range of analog and digital front end parameters, such as the sampling rate of the digital-to-analog converter, oversampling ratio and carrier frequency. These parameters are only bounded by the hardware specifications of the analog front end. By doing so, we can optimize spectrum use by changing channel and bandwidth adaptively.

Our proposed implementation (Figure 2) first reads the PSDU (physical layer service data unit) data from memory, which can be a FIFO coming from the host command processor or an internal memory emulated by the MAC processor. Next, the packet data is used to form the PPDU (PLCP protocol data unit) packet, where we can choose to change the start-of-frame delimiter, add a cyclic redundancy check (CRC), change the CRC algorithm or polynomial, and change the maximum length of the packet. The symbol-to-chip mapping block allows us to modify the chipping sequence, which could allow for non-standard spreading codes for, e.g., strengthened privacy. The modulation is performed using OQPSK with configurable constellation and pulse shape.

In addition to the configuration parameters, it is possible to add functional blocks, such as extra filters for multi-band spectrum aggregation or digital mixers for multi-channel operation. As mentioned before, this requires modifications to the LabVIEW FPGA program and a recompilation of the code. While this is still relatively user friendly because of the graphical user interface, it involves some more advanced PHY layer knowledge, especially when targeting advanced PHY functionality. Gradually, more PHY functional blocks will become available.

3.2.2 FPGA PHY receiver

Figure 3 shows the current receiver implementation of IEEE 802.15.4, which is again a library of parameters and functional blocks. Table 2 lists all the parameters of the receiver, including both standard compliant modes and extended configuration modes, similarly as for the transmitter. The receiver first downconverts the signal, where a large range of possible sampling frequencies and carrier frequencies is possible, only constrained by the RF front end and analog-to-digital converter. The baseband samples are then used to calculate the received signal strength indicator (RSSI), which is reported to the MAC processor for evaluating the clear channel assessment (CCA). The CCA threshold is parameterized and can hence be controlled by the command processor or embedded controller. The receive datapath then continues first with carrier frequency offset (CFO) compensation, after which the signal is MSK demodulated. The receiver next correlates the demodulated chips with the chipping sequences to produce the symbols. The system subsequently synchronizes on the start of frame delimiter and extracts the packet. Again, these blocks should be configured to be compatible with the transmitter. For example, if the CRC is changed on the transmitter side, it must be made compliant on the receiver side. The resulting packets are then transferred to either the PC or the embedded system.

Similar to the transmitter, it is also possible to replace or upgrade the available functional blocks, e.g., to add other

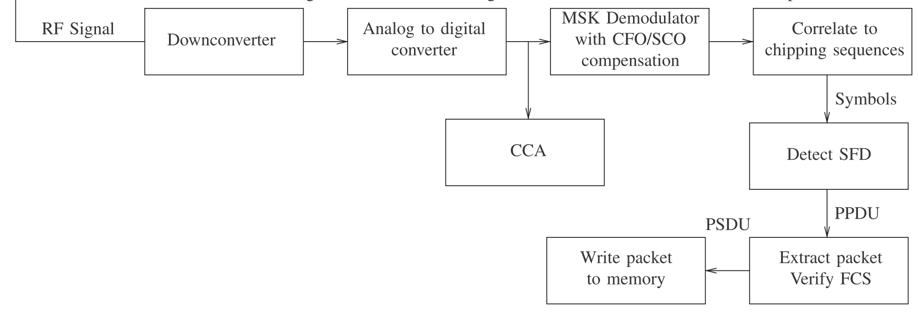
Analog waveform
Digital waveform
Demodulated chips
RF Signal
Downconverter
Analog to digital converter
MSK Demodulator with CFO/SCO compensation
Correlate to chipping sequences
Symbols
CCA
Detect SFD
PPDU
PSDU
Write packet to memory
Extract packet Verify FCS

Figure 3 LabVIEW FPGA PHY RX. Block diagram of the PHY receiver.

Table 2 FPGA PHY RX parameters

	Standard parameters	Extended parameters
Downconvert	Channel number	Any frequency
Sampling		Sampling rate
CCA	Limited threshold range	Any threshold
OQPSK demod		Modulation type
Chip to symbol		Chipping sequence
Sync		Other SFD
Extract packet		Change CRC
		Maximum packet length
Write to memory		Change destination

These are all parameters that can be changed on the CLAWS receiver. The first column corresponds to the blocks of Figure 3. If a cell is empty, this means nothing can be changed to this block.

synchronization or CFO compensation schemes, implement other OQPSK demodulators or add totally novel blocks such as filters for multi-channel bonding, spectrum sensing, or digital mixers for multi-channel operation.

3.3 MAC processor

A programmable system has been implemented for converting high-level commands from the network stack, such as 'send a packet', into bit-level instructions for the PHY hardware blocks. This system consists of a small programmable processor dedicated to running the lower MAC tasks. Therefore, it is easy to get sufficiently deterministic real-time performance without impacting the application layers, as would be the case if this was done on the main CPU. The program memory can be updated on-the-fly by the host, which allows for easy switching of the MAC algorithms in use. The interface to the main CPU is by means of a shared memory that both the MAC and main CPU can write to and read from.

To complete the picture, we will explain the tasks performed by the MAC processor when CLAWS is receiving a packet:

- PHY receives a start-of-frame delimiter. The PHY sends an interrupt to the MAC processor, which starts the receive routine.
- PHY receives data bytes. They are transferred to the MAC processor, which stores them in the shared memory. A copy of the packet header is also kept in the local memory of the MAC processor.
- Packet is finished. All bytes have been received, and the MAC processor signals to the main CPU that a complete packet is available in the shared memory. When frame check sequence (FCS) checking is enabled, the signaling is only done if the FCS is correct.
- MAC processor checks the header of the packet to see if an acknowledgement is required. If not, the MAC is now idle again. If it is enabled, it continues with the next steps.
- Start PHY TX. The packet length is three and a FCS should be appended.
- Deliver bytes for transmission. When the PHY asks for a byte to transmit, the MAC will deliver it.
- Transmission ended. The ACK has been sent and the MAC is now idle. An event could be sent to the main CPU if desired.

3.4 Networking and OS layer

It is our strong belief that to fully realize the promises of SDR as a wireless innovation platform also for the networking community, it is required to develop fully embedded systems including software and tools common to the networking community. Most sensor nodes consist of a processor connected to a radio chip and one or more sensors or actuators. To emulate this, our IEEE 802.15.4 radio is complemented with a softcore processor. The chosen softcore processor is the Xilinx Microblaze [19], but alternatives exists, for example, OpenRISC. The Microblaze processor can be programmed in C, which most sensor network researchers will be familiar with since this language is also used for programming the processor in most common off-the-shelf sensor nodes (often a MSP430 or AVR). Furthermore, many sensor node platforms use an operating system to allow smooth development. The standard operating system, Contiki OS, was ported to our platform.

Of course, most real-world applications require more than a system that is just pingable, and many benefits arise from selecting Contiki OS for our platform. First, this allows experimentation with the variation of network layer adaptations that have already been developed for Contiki OS. These innovations can be validated not only on a network of SDRs, but also on a heterogeneous network consisting of off-the-shelf sensor nodes and SDRs. Porting Contiki OS to CLAWS makes operating the system turn-key. Indeed, if a suitable network border router is nearby, one can start CLAWS and obtain 6LoWPAN connectivity immediately, relying on the functionality provided by Contiki OS. Contiki OS also helps with further development by providing a full IPv6 stack that for example allows transmitting sensor values over UDP to a computer or even an embedded webserver. The MAC processor we proposed connects the PHY hardware blocks to the Contiki-based system running on the main processor. Since this MAC processor is separated from Contiki OS, other operating systems, or if needed bare-metal networking code, could be ported to CLAWS by extending the MAC processor where needed. This means that the architecture is not limited to Contiki OS only, yet we believe Contiki OS is a good starting point to tap into the networking communities.

3.5 Host control and PHY benchmarking framework

It is understood that not every experiment will require real-time control by the embedded processors, and link or PHY layer experiments, as typically carried out by the PHY community, should still be enabled. To meet this demand, a command interface has been developed. This interface with the host computer is written in LabVIEW. With this interface, the host can transmit and receive packets and control the parameters of the datapath via the PCIe interface. Since the computationally intensive DSP PHY implementation is still executed on the FPGA, the load for the computer is very low. The interface is a standard TCP/IP connection using a command/response protocol, as seen on Figure 1.

Exploiting this interface, we have developed a benchmarking framework that can measure bit and packet error rates over arbitrary communication channels. It has been used to compare our PHY implementation against other off-the-shelf PHY solutions. This command/response protocol is generic and can also be implemented on other devices, as has been done for the Zigduino to enable a PHY comparison framework.

4 Functional validation and performance results

For our performance evaluation, we will first compare our radio design with off-the-shelf implementations to prove that similar performance can be obtained and that the radio can form a network with those implementations. For PHY testing, we use our PHY benchmarking framework with the host command/control protocol. For network testing, we use the CLAWS nodes with the Contiki OS embedded control in a network. These results will also show that it is very difficult to compare performance of different hardware solutions in exactly the same scenarios, due to the differences in hardware (independent from the functionality). This motivates the need for a universal radio, which can emulate off-the-shelf performance and can hence compare various algorithmic improvements at the PHY/MAC layer, independent of pure hardware specs.

4.1 Experimental setup

For our PHY experiments, the setup as shown in Figure 4 was used, consisting of the proposed command/control PHY layer benchmarking framework. To ensure PHY layer measurements that can be repeated independently of ambient interference, we connect all nodes with a coaxial cable. In addition to the transmitter and receiver, an external noise generator is added. The noise generator, which is implemented on an additional USRP, is used to create an artificially high noise floor, which allows emulating various signal-to-noise ratios (SNR) without being impacted by receiver sensitivity limits.

We have used commercially available Zigduino nodes [21] to benchmark our implementation. The Zigduino is a board with an Atmel CPU and 802.15.4 radio system-on-chip (ATMEGA128RFA1). These Zigduino nodes run software developed for this project that implements the same command protocol as CLAWS. Since the Zigduino only has a serial port (USB-based, provided via a FTDI UART chip), a TCP-to-serial-port proxy, ser2net, is used. All experiments have been done on channel 26, which corresponds to a frequency of 2.48 MHz. Transmit power measurements have been performed over a 5 MHz bandwidth. One of the advantages of our system is that transmit power control is very linear and over a wide range. The power control of the off-the-shelf radio we used is rather non-linear, requiring the measurements using the Zigduino as transmitter to be done in different output power ranges with attenuators added and removed manually, illustrating how hard it can be to compare various hardware platforms with the same functionality over the same scenario (SNR range in this case).

4.2 PHY layer SDR performance

In Figure 5, we plot the bit error rate (BER) performance comparison between our CLAWS transceiver and the off-the-shelf Zigduino transceiver. It can be seen that a similar BER behavior can be achieved in terms of receiver PHY performance. With similar BER performance, we mean that for high SNR, a similar and low BER is achieved. However, in terms of transmitter PHY performance, we see a 4 dB difference; this is mainly due to signal power measurement errors (as noted before, it was not possible to

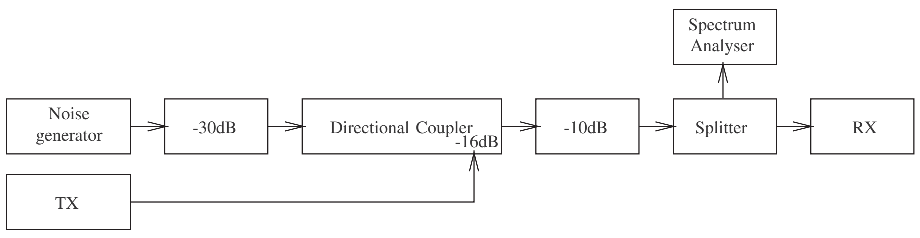

Figure 4 Experimental setup. Setup used for the BER/SNR experimental results.

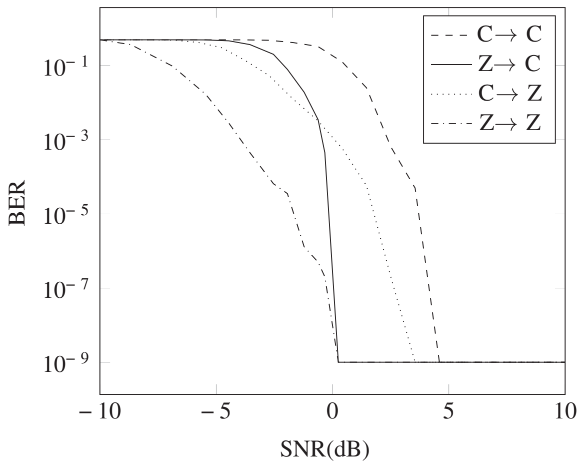

Figure 5 BER performance comparison. Bit error rate comparison between CLAWS (C) and Zigduino (Z).

test the transmitters in the same power range due to limited output power of the Zigduino). In general, a 4 dB performance difference is small enough to test all higher layer improvements. We can conclude that the CLAWS transceiver performance is within the acceptable range of the Zigduino performance, although not identical.

4.3 Network layer performance

To evaluate the full system performance, we will measure round-trip delay which is one of the most important parameters. For these measurements, an IPv6 network is set up using a Zigduino as border router (see Figure 6). The border router is connected to the host computer via the USB serial port provided by the Zigduino. The USB connection is configured for a bitrate of 1 Mbps. The network was set up to use RPL (IPv6 Routing Protocol for Low power and Lossy Networks), which is a standard routing protocol designed for low-power wireless networking that is often employed for sensor networks. Of course, a network with a single node is not suitable for evaluating the routing aspects of RPL. This is not the point of this test; we mainly want to demonstrate interoperability with the standard networking technology. To measure the delay, ping packets (ICMP Echo, 37 bytes payload) are sent to and received from CLAWS or Zigduino nodes.

Figure 7 shows a timeline of the events that happen during the ping test. First, the packet has to be processed by the USB connection and border router ①. Next the packet is transmitted over the IEEE 802.15.4 over-the-air channel ②. In the third step, the target device (CLAWS or Zigduino) executes the networking code to produce a reply packet ③. The fourth step is transmission of the reply packet back to the border router ④. Finally, the border router has to process the packet and forward it to the host PC ⑤. The complete round-trip time to the Zigduino is 12.9 ms, while the CLAWS system clocks in at 11.9 ms. Again, we see that the performance of the Zigduino and CLAWS is almost identical. The processing phase of CLAWS is slightly shorter due to the faster CPU. One should note that the radio air-time is longer than what one would expect for 37-byte-long ping packets. This is caused by 6LoWPAN/ICMP protocol overhead,

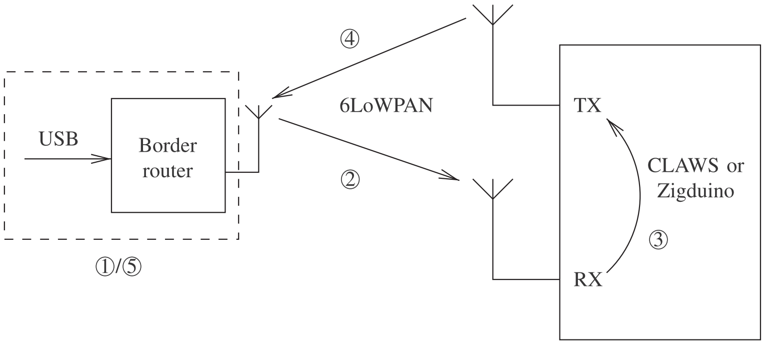

Figure 6 Network testing setup. Setup used for the network latency tests.

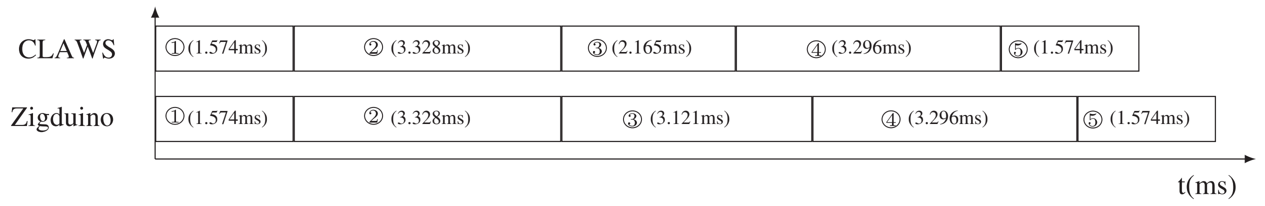

Figure 7 Ping timeline. Measured round-trip time for different targets split into the subparts from Figure 6.

and the downlink packet is 104 bytes, while the uplink packet is 103 bytes long. To demonstrate that RPL routing is also working, we have built a setup consisting of two CLAWS nodes and the border router. Due to the extra hop, the round-trip time to the last node was measured as 19.8 ms; this is not shown in Figure 7.

While Contiki OS in the configuration we used for these tests does not use IEEE 802.15.4 ACKs, we have observed the minimum time required by our system to send an ACK. This was significantly faster (8 μs) than allowed by the standard (192 μs) and serves as an indicator that real-time performance can be achieved with this architecture, potentially for much more demanding protocols, like IEEE 802.11.

5 Cross-layer design and benchmarking

From the previous sections, it is clear that the CLAWS performance on both the PHY and MAC layer is comparable to off-the-shelf nodes, making it suited for testing changes in both layers and comparing with the default functionality (on the same hardware, avoiding calibration errors). As indicated in the introduction, we believe most groundbreaking research requires a cross-layer approach. Due to the way CLAWS is built, it is perfectly suited for implementing innovative solutions requiring changes to be made to more than one layer, leveraging also on tools commonly known to the PHY or networking communities (SDR and Contiki OS). Below, we give an overview of existing cross-layer designs that could benefit from CLAWS. Next, a very simple case study is presented and analyzed. The main goal of this case study is to show the cross-layer experimentation potential of our platform.

5.1 Cross-layer design on CLAWS

Various cross-layer design approaches, proposed in literature over the past decade, could easily be verified in CLAWS. In [22], a very scalable networking solution is proposed that allows Wi-Fi networks to adapt to the available spectrum precisely, tuning center frequency and bandwidth. The authors however note that due to the capacity limit of the general-purpose processor, they cannot run their algorithm in real-time and offline decoding is necessary. Such schemes could be implemented on the CLAWS architecture elegantly and would only require some changes to the Contiki protocol layers,

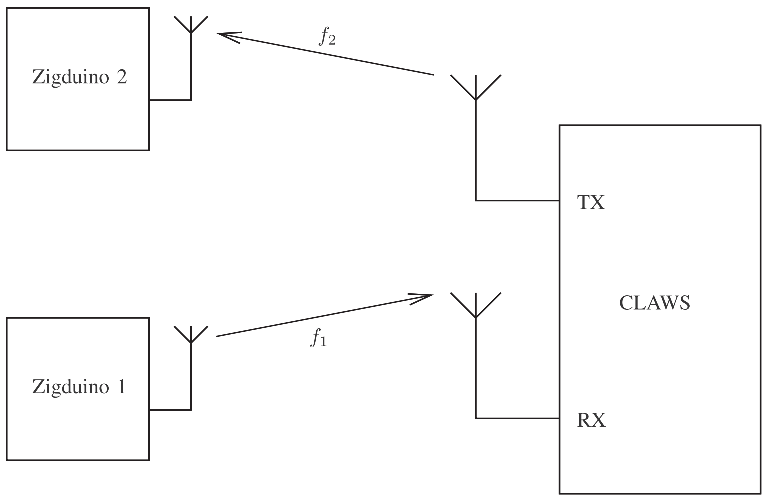

Figure 8 Relay design architecture. Overview of the cross-layer experiment.

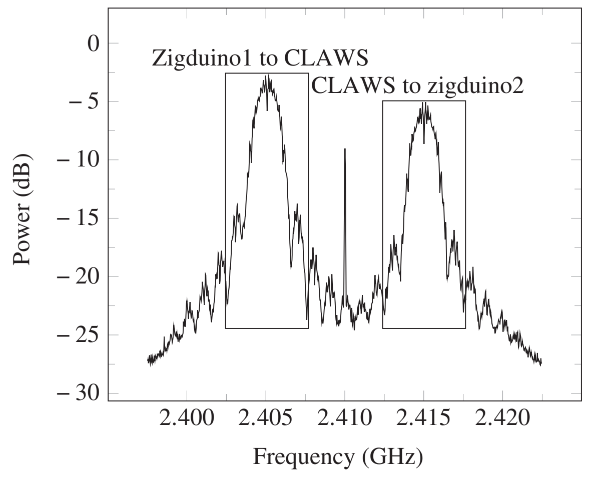

Figure 9 Relay design spectrum. The spectrogram shows channels 11 and 13 being used.

taking advantage of the extended parameter set exported to the higher layers.

Alternatively, a large amount of multi-channel MAC protocols have been proposed in the past, but real-time validation has up to now been challenging as fast enough tuning of the PHY channel has been difficult to achieve using off-the-shelf chipsets [23]. With CLAWS, leveraging the wide bandwidth of the RF front end used, it becomes possible to implement multi-channel MAC protocols with direct digital synthesis frequency shifting only. By adding a simple numerically controlled oscillator to the PHY, CLAWS can receive on various channels simultaneously or can receive on channel A and transmit on channel B with virtually no channel switching delays (just some clock cycles for the digital mixer). This is powerful as it allows multi-channel MAC prototyping beyond current radio channel switching limitations. Of course, if this level of performance is not needed, it is also possible to reconfigure the hardware phase locked loop (PLL). In this case, the PLL will need to relock after every frequency change, which reduces throughput and increases latency.

Tytgat et al. [24], for example, argue that no single channel is optimal for a large IEEE 802.15.4 network, and ideally different channels are selected for each packet reception (Receiver Defined Transmission). Performance evaluation of the required channel sensing methods and channel selection methods is very challenging using off-the-shelf IEEE 802.15.4 radios, but can be implemented on CLAWS very elegantly. One could perform the implementation in user space, by just controlling the center frequency parameter or by adding a digital mixer in the PHY layer that does the channel switching instantly. It is even possible to simultaneously use different transmit and receive frequencies. In the next section, both approaches are discussed and evaluated.

5.2 Cross-layer design case study

To show the merit of our design a relevant cross-layer design that involves receiving a packet and transmitting it on an adjacent channel was benchmarked in a multi-hop network. Using this multi-channel and multi-node implementation, we have conducted some experiments to show the benefits. A relay node was designed that receives on channel 11 and transmits on channel 13. First, the PHY was changed to allow shifting the receive and transmit frequencies using numerically controlled oscillators. For this, a novel functional block, i.e., digital shifter, was added to the PHY. This approach allows the receiver and transmitter to work simultaneously on different channels in full duplex. Off-the-shelf nodes can only transmit and receive in half-duplex. Second, changes were made to the MAC layer to allow relaying received traffic on the input frequency to the output frequency. For this, the MAC processor was changed to allow transmission and reception of packet data at the same time. This is an adaptation that is not specified by the 802.15.4 standard. Finally, the networking layer was configured to forward received packets via the second channel. Because of the small scale of the experiment, Contiki was configured to route the traffic in a static way; this meant setting the routing tables accordingly. The necessary changes on all three layers could be implemented in 15 min using the CLAWS design.

To verify operation, we have used the benchmarking framework, as explained in Section 3.5. One Zigduino node was configured to transmit on channel 11 and a second one to receive on channel 13. The setup is shown in Figure 8. As seen in Figure 9, the system is able to receive data on the first frequency while transmitting on the second. The same multi-channel relay could be set up by implementing changes at the MAC layer only,

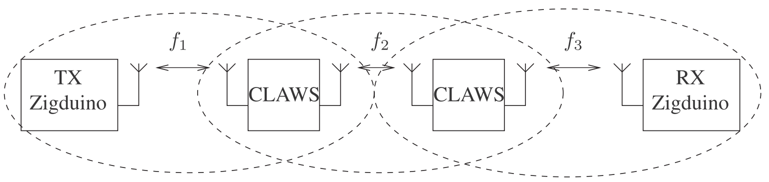

Figure 10 Multi-channel throughput setup. Setup used to benchmark the cross-layer multi-channel implementation.

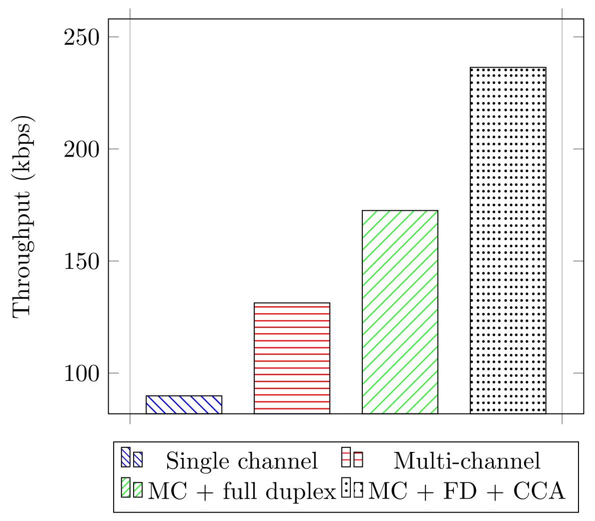

Figure 11 Throughput comparison. Comparison of different multi-channel implementations in terms of throughput.

i.e., upgrading the MAC to allow for dynamic control of the PHY center frequency. The latter approach requires however retuning the analog PLL, which involves a delay compared to the more advanced full duplex approach with digital mixing. Both can however be realized in CLAWS very efficiently and flexibly, and it is up to the protocol designer to decide what is necessary for benchmarking the protocol. When implementing the proposed multi-channel relay on off-the-shelf Zigduino nodes, of course, only the latter approach is possible, and the MAC design becomes limited by channel switching delays or hard constraints imposed by the hardware.

To further test this simple case study, we have carried out some throughput experiments. The setup is shown in Figure 10. The left Zigduino transmits packets whenever the medium is free to the first CLAWS node. The packets are then relayed by this first CLAWS node to the second one. This second CLAWS node again relays the packets to the right Zigduino. As can be seen from Figure 10, all nodes are only in range of their neighbors, making multi-hop the only possible path. Three different multi-channel optimizations are benchmarked in terms of throughput against the single-channel, standard compliant implementation. The results are shown in Figure 11. In the single channel case, frequencies f_2 and f_3 from Figure 10 are equal to frequency f_1. In this case, the throughput decreases a lot because nodes need to wait for the neighboring nodes to stop transmitting. The first improvement enables multiple frequencies for the hops, and a 50% higher throughput is achieved. The second improvement enables full duplex on multiple frequencies; for this, minor changes were needed to the software running on the MAC processor. This second improvement reaches a throughput of around 171 kbps. The 802.15.4 standard mandates a sensing period to determine if the channel is idle. Since in this case we know that no collision can occur, we disable this sensing period. The improvement afforded by disabling the nodes' CCA increases the throughput to 95% of the theoretical throughput.

This simple case study shows two things: First, it shows that it is possible to make changes to multiple layers on CLAWS and benchmark them in real-time. Second, it shows that small cross-layer improvements can have huge benefits.

6 Conclusions

This paper describes a cross-layer, flexible SDR solution and benchmarking framework, using commercially available SDR technology. The developed system can communicate with off-the-shelf sensor nodes and allows for tuning of both PHY and higher protocol layers. This architecture is provided to the open source community [25], in order to become a framework for validating and benchmarking wireless cross-layer innovations. Future work involves adding novel functional blocks enabling improved PHY performance, as well as more configuration options or richer protocol development. Also, other technologies such as IEEE 802.11 will be selected and added to the framework. Finally, larger scale benchmarking experiments will be conducted, testing the SDR in large networking configurations, such as the one available in the FP7 project CREW [26].

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

BVdB was responsible for the VHDL programming, MAC processor, and Contiki integration. TV was responsible for the LabVIEW programming. Both cooperated on the experiments and writing of the paper. MV gave useful feedback during the project phase. SP is the promoter of both TV and BVdB, and main primary investigator of the testbed infrastructure for CLAWS, driving the research and writing part of the motivation and state-of-the-art sections. All authors read and approved the final manuscript.

Acknowledgements

This research has been partially funded by the IWT SBO project SINS and a Hercules infrastructure grant. The first author is funded by the Fund for Scientific Research, Flanders (F.W.O.-Vlaanderen).

Received: 27 February 2014 Accepted: 21 October 2014
Published: 12 November 2014

References

1. S Pollin, M Timmers, L Van Der Perre, *Software Defined Radios: from Smart(er) to Cognitive*. (Springer, Dordrecht, 2011)
2. Dresden sets up 5G communications research lab. http://www.eetimes.com/document.asp?doc_id=1266564. Accessed 15 Oct 2014
3. JI Choi, M Jain, K Srinivasan, P Levis, S Katti, in *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*. Achieving single channel, full duplex wireless communication (ACM New York, 2010), pp. 1–12
4. T Schmid, O Sekkat, MB Srivastava, in *Proceedings of the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*. An experimental study of network

performance impact of increased latency in software defined radios (ACM New York, 2007), pp. 59–66

5. G Nychis, T Hottelier, Z Yang, S Seshan, P Steenkiste, in *NSDI,* Enabling MAC protocol implementations on software-defined radios, vol. 9 (Boston, Massachusetts, 22–29 Apr 2009), pp. 91–105
6. A Puschmann, MA Kalil, A Mitschele-Thiel, A flexible CSMA based MAC protocol for software defined radios. Frequenz **6**(9-10), 261–268 (2012)
7. B Bloessl, C Leitner, F Dressler, C Sommer, *A GNU Radio-based IEEE 802.15. 4 testbed. 12. GI/ITG FACHGESPRÄCH SENSORNETZE, 37* (Cottbus, Germany, 2013)
8. WikiStart - GNU Radio. http://www.gnuradio.org. Accessed 15 Oct 2014
9. C Hunter, J Camp, P Murphy, A Sabharwal, C Dick, in *Fortieth Asilomar Conference on Signals, Systems and Computers, 2006. ACSSC'06*. A flexible framework for wireless medium access protocols (IEEE, Pacific Grove, California, 29 Oct – 1 Nov 2006), pp. 2046–2050
10. MC Ng, Airblue: a highly-configurable FPGA-based platform for wireless network research. PhD thesis, Massachusetts Institute of Technology (2011)
11. D Raychaudhuri, M Ott, I Secker, in *First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005 Tridentcom 2005,* Orbit radio grid testbed for evaluation of next-generation wireless network protocols (IEEE, Trento, Italy, 23–25 Feb 2005), pp. 308–309
12. w-iLab.t (iMinds) | CREW project. http://www.crew-project.eu/wilabt. Accessed 15 Oct 2014
13. J Ansari, X Zhang, A Achtzehn, M Petrova, P Mahonen, in *Wireless Communications and Networking Conference (WCNC), 2011 IEEE,* A flexible MAC development framework for cognitive radio systems, (Cancun Quintana-Roo, Mexico, 28–31 Mar 2011), pp. 156–161
14. D van den Akker, C Blondia, in *2012 21st International Conference on Computer Communications and Networks (ICCCN),* MultiMAC: a multiple MAC network stack architecture for TinyOS, (IEEE, Munich, Germany, 30 Jul – 2 Aug 2012), pp. 1–5
15. FLAVIA Home page. http://www.ict-flavia.eu/. Accessed 15 Oct 2014
16. TinyOS Home Page. http://www.tinyos.net. Accessed 15 Oct 2014
17. Contiki: the open source operating system for the Internet of Things. http://www.contiki-os.org. Accessed 15 Oct 2014
18. LabVIEW FPGA Module - National Instruments. http://www.ni.com/labview/fpga/. Accessed 15 Oct 2014
19. MicroBlaze soft processor. http://www.xilinx.com/tools/microblaze.htm. Accessed 15 Oct 2014
20. NI USRP-2942R - National Instruments. http://sine.ni.com/nips/cds/view/p/lang/nl/nid/212434. Accessed 15 Oct 2014
21. Zigduino r1 manual. http://wiki.logos-electro.com/zigduino-manual. Accessed 15 Oct 2014
22. S Yun, D Kim, L Qiu, in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking,* Fine-grained spectrum adaptation in WiFi networks, (ACM, 2013), pp. 327–338
23. H-SW So, G Nguyen, J Walrand, in *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking,* Practical synchronization techniques for multi-channel MAC, (ACM New York, 2006), pp. 134–145
24. L Tytgat, O Yaron, S Pollin, I Moerman, P Demeester, Avoiding collisions between IEEE 802.11 and IEEE 802.15. 4 through coexistence aware clear channel assessment. EURASIP J. Wireless Commun. Netw. **2012**(1), 1–15 (2012)
25. CLAWS Home Page. http://www.claws.be/. Accessed 15 Oct 2014
26. Project Overview | CREW project. http://www.crew-project.eu. Accessed 15 Oct 2014

doi:10.1186/1687-1499-2014-187
Cite this article as: Van den Bergh *et al.*: **CLAWS: Cross-Layer Adaptable Wireless System enabling full cross-layer experimentation on real-time software-defined 802.15.4.** *EURASIP Journal on Wireless Communications and Networking* 2014 **2014**:187.