A GNU Radio-based IEEE 802.15.4 Testbed

Bastian Bloessl, Christoph Leitner, Falko Dressler, and Christoph Sommer
Computer and Communication Systems, Institute of Computer Science, University of Innsbruck, Austria
{bloessl,christoph.leitner,dressler,sommer}@ccs-labs.org

***Abstract*—We present a Software Defined Radio (SDR) based IEEE 802.15.4 transceiver testbed for GNU Radio. Our testbed is Open Source and fully interoperable with off-the-shelf TelosB sensor motes and the Contiki sensor mote operating system, from the physical layer to the network stack. The testbed can be setup and configured easily via a graphical user interface and applications can interface with the SDR using TCP sockets. Furthermore, the SDR is able to log traffic in PCAP format to investigate networks with common software like Wireshark.**

We believe the main applications of the transceiver to be two-fold. First, the communication stack has a modular, layered structure, which allows for rapid prototyping, is educational, and is easy to grasp, lowering the steep learning curve that SDRs typically have. Secondly, the integration of a network stack in GNU Radio pushes interoperability from the physical to the network and application layer and thus enables the investigations of higher layer metrics with SDRs.

I. Introduction

Due to their multitude of possible applications, Wireless Sensor Networks (WSNs) are still an active research topic. The application scenarios for these networks range from medical monitoring for drug dispense, environmental monitoring for precision agriculture, over industrial real-time monitoring of production systems, to event detection systems like burglar alarms. Since its release in 2003, IEEE 802.15.4 [1] emerged as the de-facto standard for these networks, enabling higher layer standards like WirelessHart, ZigBee or 6LoWPAN.

The success of the standard is also visible in the fact that, today, there are twelve different physical layers proposed [2]. Starting with an O-QPSK physical layer with channels in the 700 MHz to 900 MHz and 2.4GHz ISM-bands, IEEE 802.15.4 networks are now even considered as secondary users on locally unoccupied parts of the spectrum like TV white space.

For that reason, system designers not only have to decide between different frequency bands, but also between completely different wireless technologies, like O-QPSK, GFSK, or OFDM, i.e., selecting from a spectrum that ranges from single to multi-carrier systems. It is therefore crucial to be able to compare and evaluate the performance of these different physical layer technologies. Furthermore, as WSNs are extremely application specific, it is important not to be limited to physical layer metrics like Signal to Noise Ratio (SNR) and packet loss, but to have the opportunity to compare application layer behavior and metrics like true goodput.

Experimentation is a valuable instrument that can complement analytic and simulative performance evaluation of wireless communication systems to considerably increase the confidence in the results.

Currently, there are two general approaches to conduct experimental research. First, one can rely on off-the-shelf sensor motes. This approach is easy to realize and, due to typically relatively cheap motes, a larger network can be investigated with moderate costs. The drawback of using real motes is, however, that the insights are limited to the information that the transceiver chip provides. Off-the-shelf sensor mote testbeds are, therefore, well-suited to investigate application layer metrics, but might lack the possibility to explain some of the effects, as advanced metrics are not accessible. For example, it can be hard to determine if outage occurred due to interference or due to noise, as ordinary transceiver chips do not provide any information about packets that could not be decoded. Another potential drawback is that off-the-shelf motes can not be used to investigate new physical layer strategies. Furthermore, for some of the proposed physical layers, there are no consumer devices available yet.

A second approach for conducting experimental research is the use of Software Defined Radios (SDRs), where signal processing is done in software instead of being hidden inside a transceiver chip. An SDR system consists of a software framework for real-time signal processing and a hardware RF frontend to send and receive the signal. With such a reprogrammable system the user has full control over all signal processing steps. The drawback of SDRs is that they add cost and complexity. SDR-based systems are often not accessible for non-experts in signal processing. Furthermore, these systems are often limited to physical layer implementations only and, thus, are not interoperable with real sensor motes on higher layers, i.e., the SDRs can not become nodes in a WSN.

We bridge this gap by providing an SDR based IEEE 802.15.4 testbed that provides a fully interoperable network stack. The testbed is implemented based on GNU Radio and implements the communication stack from the physical up to the network layer, where applications can be attached easily. As network layer, we choose the Rime stack [3]. Rime is a modular, lightweight network stack, which is part of the Contiki operating system. Contiki [4], in turn, is a state-of-the-art operating system for research in WSNs. With the help of the Rime stack, the SDRs can be easily integrated into a WSN, or form a heterogeneous network consisting of sensor motes and SDRs.

We make all GNU Radio code, a demo application, and a Contiki firmware available as Open Source software in the hope that it might be useful for others.[1]

[1] All source code can be downloaded from http://www.ccs-labs.org/software/.

II. Related Work

The O-QPSK physical layer of the IEEE 802.15.4 standard was first implemented by Thomas Schmid in 2006 [5]. This implementation featured separated receive and transmit chains and was verified to work with Crossbow MicaZ motes.

This implementation provided the base for several further research projects. It was used to study properties of the physical layer and the applicability of SDRs to conduct wireless research in general.

The potential of SDR based testbeds for wireless research in sensor networks was already realized by Ali et al. [6], urging the community to move from simulations to SDR based prototypes. In [7], the authors go one step further and discuss the idea of *Cognitive Radio Sensor Networks*. These networks exploit the flexibility of SDRs by applying cognitive radio strategies to sensor networks. An actual testbed for wireless research is presented in [8], where the authors prototype protocols in SDRs and on real hardware, but focus only on the MAC layer.

In all SDR implementations latency is a crucial factor, as the standard mandates strict maximum response times. For that reason, the latency of the GNU Radio implementation is studied by Thomas Schmid et al. in [9] by means of Round Trip Time (RTT) measurements.

Using SDRs connected via USB 2.0, the RTT of the SDR was an order of magnitude higher than the RTT of MicaZ motes. Furthermore, the RTT of the SDR suffered from high variances, whereas the RTT of the MicaZ motes was very deterministic. Given these results, the latency bounds that the IEEE 802.15.4 standard mandates could not be met.

The more recent USRP2 SDRs are connected via Gigabit Ethernet. This, together with the advances of GNU Radio, most notably the Vectorized Library of Kernels (VOLK) [10], should lower latency considerably. The VOLK library is part of GNU Radio and allows the use of Single Instruction Multiple Data (SIMD) operations. As SIMD instructions operate on vectors instead of scalars, the performance can be improved considerably.

In [11], the IEEE 802.15.4 implementation was further extended to support multi-channel reception with the N210 USRP SDRs. Compared to the USRP1, the N210 provides a higher bandwidth and allows to decode five adjacent channels in parallel.

III. Transceiver Architecture

Figure 1 gives an overview of the transceiver structure as exposed to GNU Radio Companion, a GUI that can be used to setup and configure signal processing flow graphs. The layered structure of the communication system can be identified clearly. In the following, we describe the individual components.

A. Physical Layer

For our tests and development of the receiver, we used USRP N210 SDRs from Ettus Research, equipped with XCVR2450 daughterboards as radio frontend. These daughterboards can operate on the 2.4 GHz ISM band in half-duplex mode.

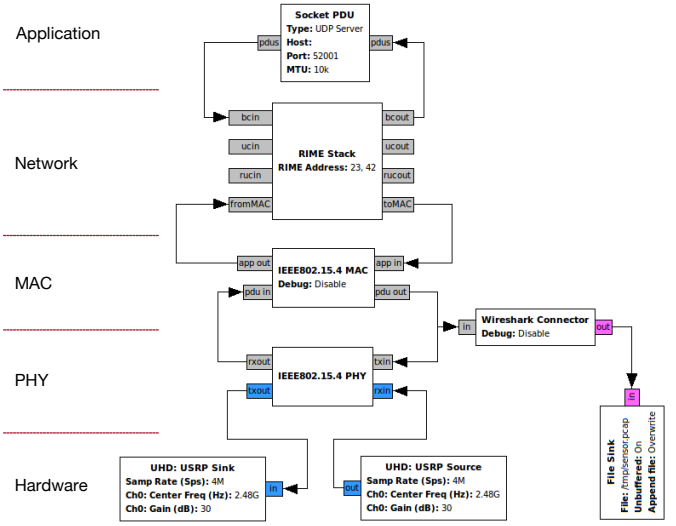
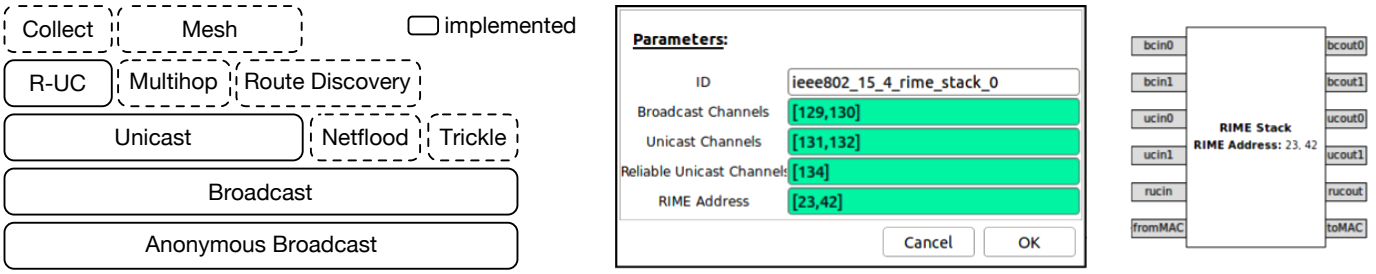


Figure 1. The modular, layered structure of the SDR transceiver as exposed to GNU Radio Companion. The physical Layer is encapsulated in a hierarchical block. The packets between the MAC and the physical layer are captured by the *Wireshark Connector* (in this case only outgoing packets to preserve clearness of the Figure).

The *USRP source* and *USRP sink* blocks in Figure 1 are interfacing this hardware. These blocks are connected to the physical layer, which is encapsulated in a hierarchical block, that hides all details of the modulation process. In GNU Radio, a hierarchical block does not implement an algorithm itself but contains another flow graph. This concept supports modularity and allows a clearer structure.

From the multitude of physical layers that are included (or, rather, are proposed to be included) in the IEEE 802.15.4 standard, currently only the O-QPSK PHY is implemented. This physical layer defines 16 channels in the 2.4 GHz band. The implementation is based on the UCLA ZigBee PHY of Thomas Schmid [5]. We extended it by porting it to version 3.7 of GNU Radio, we added GNU Radio Companion bindings in order to access the blocks in the graphical user interface, we reimplemented all python functions in C++, we changed the transmitter to GNU Radio blocks (i.e., removed custom blocks that are not needed anymore), and finally merged the separated receive and transmit chains to operate in parallel to form a transceiver system.

Creating a transceiver out of separated receive and transmit chains is not as straight forward as it might sound. Since we use half-duplex radio frontends, the USRP has to switch between send and receive mode for every packet that is sent. This happens automatically, in the sense that by default the USRP receives and switches to transmission mode if it receives samples from GNU Radio. The switch from receiving to transmitting is no problem, however the other way round is. When a packet (i.e., a burst of samples) is sent and the sample stream to the device stops, the USRP first assumes that an underflow occurred, i.e., the PC can not deliver the samples fast enough. For that reason, the device does not switch back



(a) Overview of the modular structure of the Rime stack. Each protocol makes use of its underlying primitives. Solid components have been implemented in our testbed.

(b) Configuration interface of the Rime block with ports dynamically created by the given parameter set.

(c) Resulting block instance.

Figure 2. Overview of the Rime stack and implementation in GNU Radio.

to receive mode immediately, but instead waits until a timeout occurs. The problem, however, is that when communicating with real hardware like the TelosB motes, this timeout period is too long and immediate responses like acknowledgments are missed, as the device is still in transmit mode.

To deal with that issue, we insert a tag at the start of each burst that indicates its length, i.e., the number of samples that the burst spans.[2] We added a block just before the USRP sink, which reads the length tag and, with this additional length information, is able to signal the USRP when the end of the burst is reached. Thus, the timeout before switching back to receive mode is avoided.

B. MAC Layer

As depicted in Figure 1, the MAC layer is implemented on top of the physical layer. The colors of the ports encode the type of data that is exchanged between the blocks. Gray is used to depict asynchronous messages ports, which were introduced in GNU Radio v3.6.3. Asynchronous messages allow to work packet-based, as opposed to stream-based – the mode in which most of the physical layer blocks are working in. Asynchronous messages can encapsulate arbitrary information by the use of polymorphic types. The GNU Radio developers, however, agreed on a Protocol Data Unit (PDU) format. A PDU is represented by a pair consisting of a dictionary and a character buffer. The buffer is used to store the actual data of the packet, while the dictionary can contain arbitrary metadata.

The MAC block is currently limited to the most basic functionality that enables connectivity. It encapsulates the packets from the higher layers with a valid IEEE 802.15.4 header and calculates and appends the CRC checksum. On the receive path, it does the reverse, removing the MAC header and checking whether the CRC is correct.

In particular, the MAC layer does not perform carrier sensing, but instead sends a message immediately. For that reason, the MAC layer does not yet support any other CSMA/CA functionality like backing off in time.

[2]In GNU Radio, tags can annotate specific samples of the sample stream with arbitrary information.

C. Rime Network Stack

The Rime communication stack is a lightweight network stack, designed for use in WSNs. It was implemented for Contiki by Dunkels et al. [3]. Contiki is a state-of-the-art operating system for WSNs and is, like TinyOS, heavily used and well accepted in the research community.

As depicted in Figure 2a, Rime has a modular, layered structure, where more complex connection primitives extend the underlying simpler ones to offer advanced features. The solid blocks in Figure 2a depict the connection primitives that are currently included in our SDR implementation.

The communication stack can be setup completely with a graphical user interface. Figure 2b shows the configuration interface of the Rime stack in the *GNU Radio Companion* GUI frontend. The user can open new Rime connections by adding channels numbers to the list corresponding to the desired connection type. We tweaked GNU Radio so that it is possible to dynamically create input and output message ports. This way, we can add a new pair of input and output port per connection. Figure 2c depicts the Rime block, generated by the given configuration. We see two pairs of broadcast and unicast connections (labeled bcinX, bcoutX, ucinX, and ucoutX) and a reliable unicast connection (labeled with rucin and rucout). Furthermore, we can configure the Rime address of the transceiver.

Considering unicast connections, the application has to specify the destination node on a per-packet basis, like it is the case for common UDP sockets. This is required as Rime operates connection-less. We decided to prefix the actual packet payload with the destination address of the target node in order to provide an easy to use interface to the SDR.

IV. PCAP and Wireshark Connector

To ease debugging and to allow monitoring communications in the WSN, we implemented a module that logs all transmissions in PCAP format[3], the de-facto standard format for packet dumps. PCAP is understood by all network monitoring tools like Wireshark or tcpdump. These tools also provide useful additional functionality like throughput and delay calculations.

[3]http://www.tcpdump.org/

We dump the network traffic with the *Wireshark Connector* block, which is depicted at the right hand side of Figure 1. When the block is started it writes a PCAP file header that includes global parameters like maximum packet size and the utilized technology. Every packet that is passed between MAC and physical layer is prefixed with per-packet information and logged. The per-packet header contains information about the packet size and includes a time stamp that indicates when the packet was received. The PCAP file can either be written to disk or to a Linux pipe where Wireshark can be attached to. With the help of the pipe the network can be monitored live. We utilized the Wireshark connector to get first insights into the latency of the presented transceiver. We ran the transceiver on a laptop with an Intel i5 CPU (2.6 GHz) and measured the RTTs between an SDR and a TelosB mote to be around 5 ms.

Wireshark supports ZigBee and, thus, can dissect the IEEE 802.15.4 MAC format, but not the Rime protocol headers. We therefore also implemented a dissector for Rime in LUA.

To demonstrate the capabilities of the testbed, we implemented and made available a Contiki firmware for the TelosB sensor mote platform. By default this firmware opens a broadcast connection and periodically disseminates the values of the light sensor on that connection. Furthermore, a notification is sent over that connection when the button is pressed.

It is extremely easy to connect to the flow graph with the help of the *Socket PDU* block. Connecting to the Rime connection of the flow graph and printing of the sensor values can be done with a line of shell code:

```
nc -u localhost 52001 | od -vsw2
```

As shown in Figure 3, we also provide a more visual representation of the results in a GUI that uses *matplotlib* to draw a live graph of the sensor values. Furthermore, the firmware includes a shell application where the user can connect via a serial connection over USB and dynamically open new connections and send messages on them.

V. Conclusion

We created a GNU Radio-based IEEE 802.15.4 testbed. The testbed is interoperable with real sensor motes on physical layer, and with Contiki, a state-of-the-art operating system for WSNs, on network layer. With this level of interoperability it is possible to set up a mixed network, consisting of off-the-shelf sensor motes and SDR-based transceivers. The SDR-based transceiver can be setup and configured quickly with the help of a graphical user interface. The whole communication stack, from physical to application layer, is implemented within the SDR system. Applications can be attached to the transceiver via TCP or UDP sockets.

Debugging, logging, and monitoring of the communication is aided by an option to capture all traffic in PCAP format. To further increase usability, we implemented a Wireshark dissector that parses the utilized protocols. Finally, the platform is accessible as it is Open Source.

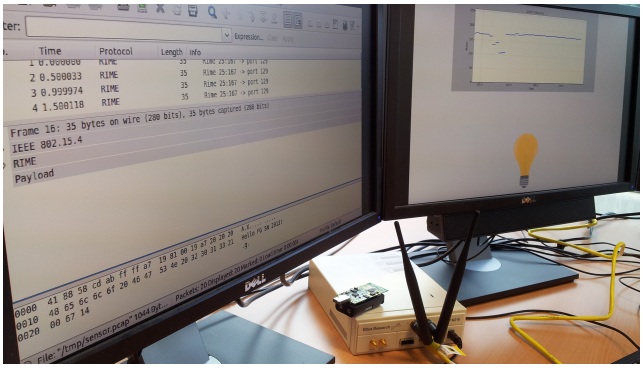


Figure 3. An example setup, consisting of a USRP N210 and a TelosB mote that periodically sends light sensor data that is displayed in the graph on the right screen. On the left screen, the whole communication is monitored in Wireshark.

This transceiver represents a proof of concept implementation that is interoperable with real sensor motes. With this as a starting point, we hope to provide a tool that facilitates rapid prototyping of new physical layers, and that allows the investigation of application layer metrics with SDRs.

References

[1] "Low-Rate Wireless Personal Area Networks (LR-WPANs)," IEEE, Std 802.15.4-2011, June 2011.

[2] C.-S. Sum, L. Lu, M.-T. Zhou, F. Kojima, and H. Harada, "Design Considerations of IEEE 802.15.4m Low-Rate WPAN in TV White Space," *IEEE Communications Magazine*, vol. 51, no. 4, pp. 74–82, April 2013.

[3] A. Dunkels, F. Österlind, and Z. He, "An Adaptive Communication Architecture for Wireless Sensor Networks," in *5th ACM Conference on Embedded Networked Sensor Systems (SenSys 2007)*. Sydney, Australia: ACM, November 2007.

[4] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *29th IEEE International Conference on Local Computer Networks (LCN 2004)*, Tampa, FL, November 2004, pp. 455–462.

[5] T. Schmid, "GNU Radio 802.15. 4 En-and Decoding," Networked & Embedded Systems Laboratory, UCLA, Technical Report TR-UCLA-NESL-200609-06, June 2006.

[6] M. Ali, U. Saif, A. Dunkels, T. Voigt, K. Römer, K. Langendoen, J. Polastre, and Z. A. Uzmi, "Medium Access Control Issues in Sensor Networks," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 36, no. 2, pp. 33–36, April 2006.

[7] O. B. Akan, O. B. Karli, and O. Ergul, "Cognitive Radio Sensor Networks," *IEEE Network*, vol. 23, no. 4, pp. 34–40, July 2009.

[8] X. Zhang, J. Ansari, L. M. A. Martinez, N. A. Linio, and P. Mähönen, "Enabling Rapid Prototyping of Reconfigurable MAC Protocols for Wireless Sensor Networks," in *IEEE Wireless Communications and Networking Conference (WCNC 2013)*. Shanghai, China: IEEE, April 2013, pp. 47–52.

[9] T. Schmid, O. Sekkat, and M. B. Srivastava, "An Experimental Study of Network Performance Impact of Increased Latency in Software Defined Radios," in *2nd ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and Characterization (WiNTECH'07)*. Montréal, Québec, Canada: ACM, September 2007, pp. 59–66.

[10] T. Rondeau, N. McCarthy, and T. O'Shea, "SIMD Programming in GNU Radio: Maintainable und User-Friendly Algorithm Optimization with VOLK," in *Conference on Communications Technologies and Software Defined Radio (SDR'12)*. Brussels, Belgium: Wireless Innovation Forum Europe, June 2012.

[11] L. Choong, "Multi-Channel IEEE 802.15.4 Packet Capture Using Software Defined Radio," Networked & Embedded Systems Laboratory, UCLA, Technical Report TR-UCLA-NESL-200904-01, April 2009.