IEEE 802.15.4 packet analysis with Wireshark and off-the-shelf hardware

Wolf-Bastian Pöttner, and Lars Wolf

Abstract—**The ability to overhear and analyse packets is essential for the development of protocols for IEEE 802.15.4-based Wireless Sensor Networks. Besides a number of commercial hardware and software offers, only very few projects put effort in developing such solutions. This demonstration shows a simple yet powerful approach to that problem. By using a T-Mote Sky sensor node with the Contiki operating system, radio packets can be overheard and then analysed in Wireshark on an attached Linux PC. By creating custom dissector components for Wireshark, new protocols can easily be analysed and problems in the network can be debugged.**

***Index Terms*—CC2420, Packet Sniffer, Contiki, Wireshark, IEEE 802.15.4**

I. Introduction

For the development of protocols, it is essential to have a good and powerful protocol analyzer at hand. For networking protocols based on IP, which are used over Ethernet, there are numerous tools to switch the Ethernet card into promiscuous mode and subsequently sniff and analyse all packets that come along.

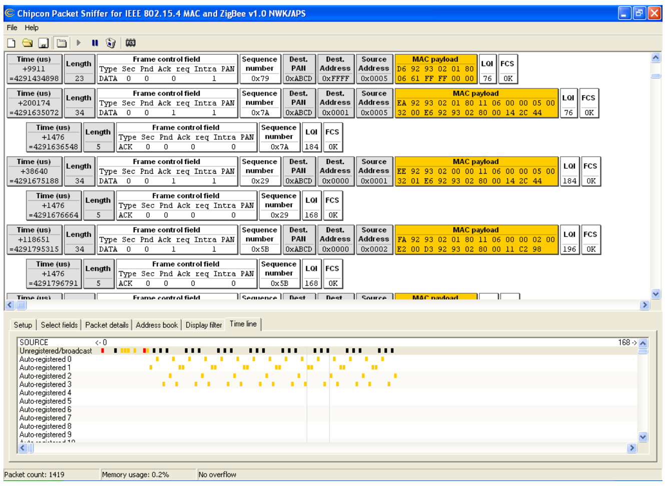
When it comes to developing low-level protocols for Wireless Sensor Networks using IEEE 802.15.4-based [1] radios for communication, the choice of such tools is limited. Simulators like Cooja [2] offer a bit of help by providing a hex-dump output of all packets that are transmitted between nodes. This can help verifying the correct creation of protocol messages but is for the simulated world only and does not scale when it comes to more complex scenarios with hundreds of packets.

For nodes in the real world, one can buy hardware from different manufacturers to sniff packets in the air and subsequently display them. However, the hardware is expensive and never available when you need it. In addition, the provided tools are usually closed source and cannot be adapted to the format of an individual protocol.

In this demo we present a very practical approach to sniffing radio packets with off-the-shelf hardware. By using the Contiki Operating System [3] on a standard T-Mote Sky [4], radio packets can be overheard and then transported to a standard Linux PC that is connected to the USB port. On the PC, the overheard packets are fed into Wireshark [5] for further protocol analysis. Custom dissectors in Wireshark allow the intuitive and efficient analysis of protocol message and thereby the diagnosis of potential problems. The contributions of this demonstration are as follows:

W. Pöttner and L. Wolf are with the Institute of Operating Systems and Computer Networks, Technische Universität Braunschweig, Germany. Email: `[poettner|wolf]@ibr.cs.tu-bs.de`

This work has been partially supported by the European Commission under the contract FP7-ICT-224282 (GINSENG).



```
Time (us)    Length  Frame control field                  Sequence  Dest.   Dest.    Source   MAC payload                            LQI  FCS
+9911                Type Sec Pnd Ack req Intra PAN       number    PAN     Address  Address  D6 92 93 02 01 80
=4291434898  23      DATA  0   0    0        1            0x79      0xABCD  0xFFFF   0x0005   06 61 FF FF 00 00                      76   OK

Time (us)    Length  Frame control field                  Sequence  Dest.   Dest.    Source   MAC payload                            LQI  FCS
+200174              Type Sec Pnd Ack req Intra PAN       number    PAN     Address  Address  EA 92 93 02 01 80 11 06 00 00 05 00
=4291635072  34      DATA  0   0    1        1            0x7A      0xABCD  0x0001   0x0005   32 00 E6 92 93 02 80 00 14 2C 44       76   OK

     Time (us)    Length  Frame control field             Sequence  LQI  FCS
     +1476                Type Sec Pnd Ack req Intra PAN  number
     =4291636548  5       ACK   0   0    0        0       0x7A      184  OK

Time (us)    Length  Frame control field                  Sequence  Dest.   Dest.    Source   MAC payload                            LQI  FCS
+38640               Type Sec Pnd Ack req Intra PAN       number    PAN     Address  Address  EE 92 93 02 00 00 11 06 00 00 05 00
=4291675188  34      DATA  0   0    1        1            0x29      0xABCD  0x0000   0x0001   32 01 E6 92 93 02 80 00 14 2C 44       184  OK

     Time (us)    Length  Frame control field             Sequence  LQI  FCS
     +1476                Type Sec Pnd Ack req Intra PAN  number
     =4291676664  5       ACK   0   0    0        0       0x29      168  OK

Time (us)    Length  Frame control field                  Sequence  Dest.   Dest.    Source   MAC payload                            LQI  FCS
+118651              Type Sec Pnd Ack req Intra PAN       number    PAN     Address  Address  FA 92 93 02 01 80 11 06 00 00 02 00
=4291795315  34      DATA  0   0    1        1            0x5B      0xABCD  0x0000   0x0002   E2 00 D3 92 93 02 80 00 11 C2 98       196  OK

     Time (us)    Length  Frame control field             Sequence  LQI  FCS
     +1476                Type Sec Pnd Ack req Intra PAN  number
     =4291796791  5       ACK   0   0    0        0       0x5B      168  OK

Packet count: 1419    Memory usage: 0.2%    No overflow
```

Figure 1. Chipcon Packet Sniffer.

- Packet sniffer implementation for Contiki 2.4
- Live sniffing interface to Wireshark with correct frame check sequence (FCS) checksum
- Implementation of a Wireshark dissector for GinMAC
- Evaluation of the performance and reliability of the presented sniffer

The remainder of this paper is organised as follows. In section II, related research efforts as well as available products are presented and discussed. Section III gives an overview over the features and possibilities of Wireshark that are relevant for this demonstration. Subsequently, section IV shows how our sniffing system works and what components it uses. Section V shows our custom dissector for Wireshark, whereas section VI shows the performance evaluation of our system. Finally, section VII lists the limitations of our approach and section VIII concludes the demonstration.

II. Related Work

Chipcon offers the RF Studio software as well as the Chipcon Packet Sniffer software that can be used together with specific evaluation boards that are also sold by Chipcon resp. Texas Instruments. The tool offers support for sniffing radio packets on specific channels. The sniffed data can be displayed and then analysed by the user as shown in Fig. 1. To support this task, the program highlights the different fields of the header and potentially the payload. The main advantage of the Chipcon Packet Sniffer is that the packets are accurately time stamped and that the tool presents a timeline. The main disadvantages are the price for the evaluation board as well as the closed source tool that handles only very limited payload formats and is not extendible.

Besides the Chipcon solution, a number of different companies also manufacture radio-sniffing hardware for IEEE 802.15.4-based networks. The hardware is usually an USB stick and comes with custom software to analyse the radio packets. Examples of such solutions are the WiSens Classic Packet Sniffer [6] and the Daintree Sensor Network Analyzer [7]. However, the disadvantages of the Chipcon solution also apply to these offers.

In addition to the commercial offers, Scheers et. al [8] have created a packet sniffer that is based on the same concept as our solution. A T-Mote Sky is flashed with the Contiki operating system. Subsequently, received radio packets are transferred over the serial port to the PC where a dedicated application saves the packets to a file in the pcap format. This file is then directly readable using Wireshark. The disadvantage of their approach is that the pcap file can only be analysed by Wireshark after the file has been completely written, so no online analysis of the network is possible. In addition, the authors write about slow speed of the serial port, which causes packets to be queued on the node. This does not only break the timing but may also cause an overload of the node.

Ban et. al [9] have created an IEEE 802.15.4 packet sniffer that uses custom hardware and is attached to the PC using an Ethernet connection. This allows remote debugging of sensor networks. However, the authors have implemented custom software that analyses the incoming packets and therefore is also limited in the number of payload packets format that can be handled.

Besides these efforts, TinyOS [10] also has tools to capture packets with standard sensor nodes and then analyse them in Wireshark. However, to the best of our knowledge, these components are currently unmaintained and do not work with the latest version of the operating system.

III. Wireshark

Wireshark is a versatile and flexible network protocol analyzer that can be extended using plugins and dissectors. Since it is open-source and freely available, it can be adapted to the needs of specific applications. Wireshark can be attached to local network interfaces, thereby overhearing incoming packets that are subsequently analysed and presented to the user. It allows to save packets into files for later analysis and to filter the displayed data. In addition, it allows colorizing the output to ease the interpretation. One specific advantage of Wireshark is that multiple dissectors can analyse the same packet. If a UDP packet is found, the payload of the packet can be passed on to the next dissector for further analysis. This is especially helpful if IPv6 packets are wrapped in IEEE 802.15.4 frames. At the moment, Wireshark supports dissecting IEEE 802.15.4, Zigbee, IPv4, IPv6 and a large number of other protocols.

IV. System Concept

The architecture of our packet sniffer is shown in Fig. 2. Packets are received on the T-Mote Sky and then transported

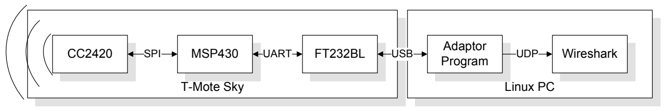
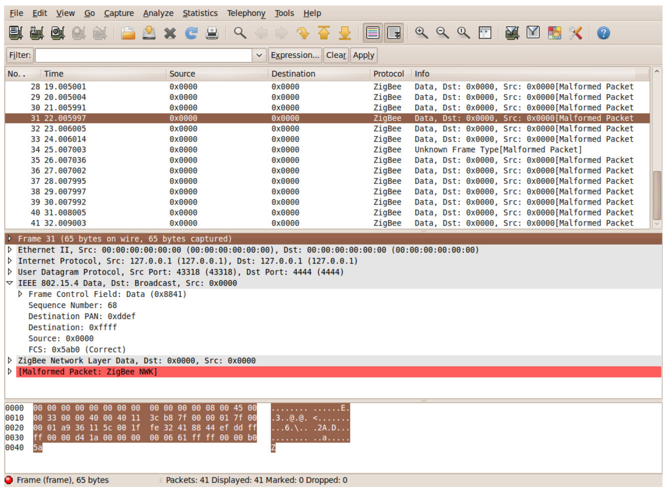


Figure 2. Packet Sniffer System Architecture.


File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression...
Clear
Apply

```
No. .  Time          Source     Destination  Protocol  Info
    28 19.005001     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    29 20.005004     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    30 21.005991     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    31 22.005997     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    32 23.006005     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    33 24.006014     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    34 25.007003     0x0000     0x0000       ZigBee    Unknown Frame Type[Malformed Packet]
    35 26.007036     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    36 27.007002     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    37 28.007995     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    38 29.007997     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    39 30.007992     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    40 31.008005     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet
    41 32.009003     0x0000     0x0000       ZigBee    Data, Dst: 0x0000, Src: 0x0000[Malformed Packet

▷ Frame 31 (65 bytes on wire, 65 bytes captured)
▷ Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▷ User Datagram Protocol, Src Port: 43318 (43318), Dst Port: 4444 (4444)
▽ IEEE 802.15.4 Data, Dst: Broadcast, Src: 0x0000
  ▷ Frame Control Field: Data (0x8841)
    Sequence Number: 68
    Destination PAN: 0xddef
    Destination: 0xffff
    Source: 0x0000
    FCS: 0x5ab0 (Correct)
▷ ZigBee Network Layer Data, Dst: 0x0000, Src: 0x0000
▷ [Malformed Packet: ZigBee NWK]

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 33 00 00 40 00 40 11  3c b8 7f 00 00 01 7f 00   .3..@.@. <.......
0020  00 01 a9 36 11 5c 00 1f  fe 32 41 88 44 ef dd ff   ...6.\.. .2A.D...
0030  ff 00 00 d4 1a 00 00 00  00 06 61 ff ff 00 00 b0   ........ ..a.....
0040  5a                                                 Z

Frame (frame), 65 bytes    Packets: 41 Displayed: 41 Marked: 0 Dropped: 0
```

Figure 3. Wireshark displaying custom radio packets.

to the PC using UART communication. There, the received packets are fed into Wireshark for further analysis.

In order to overhear radio packets with the T-Mote Sky, the CC2420 radio chip has to be configured to the specific channel and to the broadcast PAN address. Subsequently, the radio chip signals all received packets to the MSP430 processor and does not perform any filtering. The only modification to the packet performed by the radio is the autonomous verification of the CRC checksum. The RSSI, the LQI and a flag stating if the CRC was OK replace the former checksum.

Contiki, running on the MSP430 processor, uses the SPI bus to copy the incoming packet into its RAM, so that the radio is ready to receive the next packet as soon as possible. In order to transport the packet to the PC, an application running in Contiki takes the packet and adds some additional header information such as the time stamp and length information and then SLIP encodes the packet. Afterwards, the packet is send over UART.

On the PC side, a program receives the packet over UART and reverses the SLIP encoding to get the original packet. The hex-dump of the packet is then displayed on the local terminal for a quick verification of the correct function of the WSN. However, for more complex scenarios or for a large number of messages, looking at hex-dumps is time consuming and prone to errors. Therefore, the incoming packets are forwarded to a local UDP port.

By modifying only two lines in the source code, an incoming UDP datagram on a specific port can be handed over to the correct dissector. The dissector then analyses the packet and displays the result in the well-known Wireshark manner as shown in Fig. 3.

However, the Wireshark dissector assumes that the IEEE 802.15.4 frame footer, consisting of the CRC checksum, is still present in the packet. The dissector verifies the checksum against the packet and marks packets as broken if there is a mismatch. Since the CC2420 radio replaces the checksum with other information, our program running on the PC recreates the checksum in case that the CRC check succeeded. However, thereby the information about RSSI and LQI is deleted. This enables the Wireshark user to determine if a packet was received correctly on the first sight. If the CC2420 radio chip was unable to verify the CRC checksum, our program does not touch the frame footer.

V. GinMAC Wireshark Dissector

Wireshark already comes with a multitude of dissectors for various protocols. As already mentioned, this includes IEEE 802.15.4 that is very handy for the analysis of WSN network traffic. However, there are vast numbers of custom MAC protocols available, some of which use the header fields for different purposes than specified in the original standard.

One example of such custom MAC protocol is GinMAC [11]. This MAC protocol is part of the GINSENG [12] project and uses the specified packet alignment as good as possible while the semantics are different. Debugging such a custom protocol can be painful, especially when only the hex-dump of radio packets is available. Therefore, we have developed a custom dissector for GinMAC protocol messages. The dissector is integrated with Wireshark and allows users to inspect radio packets without exact knowledge of field alignments.

Another neat feature of Wireshark is that existing dissectors can be used to analyse the payload of dissected packets. Since GinMAC is able to transport IPv6 data, one can use the existing dissector to look at these packets. If the payload of the IP packets holds additional known protocols, these can again be analysed with the proper dissectors.

VI. Performance Evaluation

In order to judge the performance of the system, we have conducted measurements. However, it is important to know that all measurements presented here use the RTimer functionality of the Contiki operating system. On the T-Mote Sky, the timer produces 8192 ticks per second, which limits the accuracy and resolution of our measurements.

The first thing to measure is the throughput that can be achieved when transmitting data from the node to the PC using UART. The standard configuration of Contiki uses 115200 baud and should transport 92.16 kbit of user data per second. However, due to the maximum data rates of IEEE 802.15.4, this may not be enough to transport all received packets off the node in real-time. It is important to note that Contiki uses interrupt-triggered output for UART by default. In this setting, the function to write data to UART will buffer the data in a ring buffer. Whenever the UART transceiver signals that the next byte can be transmitted, Contiki takes the first byte off the ring buffer and puts it in the specified register.

Our aim was to increase the speed of the UART communication. In order to do this, we have taken different UART settings and measured the time it takes to transmit 65535 bytes.

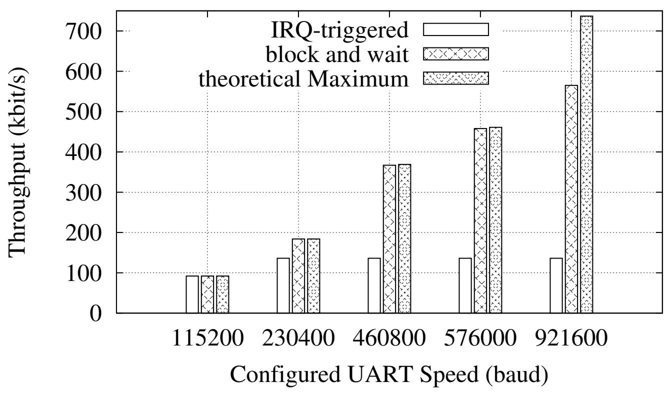


Figure 4. Achievable UART throughput of MSP430 for 3.9 MHz CPU clock.

UART Speed	Bytes Total	Bytes Corrupted
115200 baud	165148200	0
230400 baud	164834682	0
460800 baud	164492850	0
576000 baud	164492850	0
921600 baud	164492850	0

Table I
MSP430 UART reliability.

Each measurement was conducted two times, either with interrupt-triggered output enabled or disabled. Based on the determined time, we have calculated the throughputs that are presented in Fig. 4. The theoretical maximum is based on 2 bits overhead per byte (start and stop bit). We can see that there is no difference between interrupt-triggered and non interrupt-triggered UART communication in the default setting and that the theoretical maximum is reached. However, for increased speeds we can see that interrupt-triggered UART output limits the throughput to 136 kbit/s. For non-interrupt-triggered UART output the achievable throughput is almost equal to the theoretical maximum. Only for the highest baud rate supported by the hardware, the maximum of the MSP430 seems to be reached. The upper bound that we found is 565 kbit/s.

The manual of the MSP430 processor [13] implies that the probabilities for bit errors are drastically increasing with higher UART speeds, especially for a CPU clock of 3.9 Mhz. Therefore, we have measured how many bytes get corrupted while transmitting data at different baud rates. In order to do this, the node repeatedly outputs the same byte and the attached PC checks whether the data was received correctly. The test ran for one night at different speeds and the results are shown in Table I. Interestingly, not a single byte got corrupted during the UART transport. Therefore, we can safely use UART at higher speeds without the fear of drastically increased error rates. Additional measurements have to show if there is a difference between different nodes of the same type.

Another interesting parameter to measure is the time it takes to process a packet on the node. This time limits the maximum data rate at which packets can be sniffed. We first measured the time it takes to copy the packet from the radio buffer

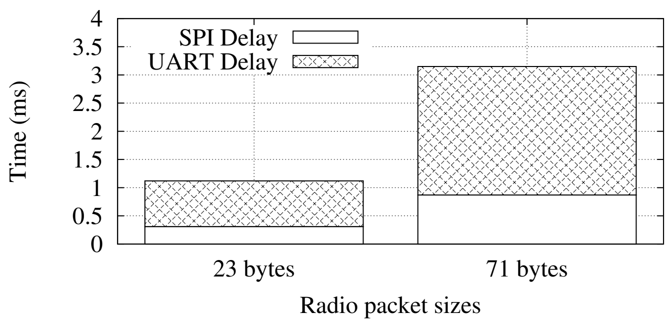


Figure 5. Time necessary to copy packets from radio to UART.

over SPI into the RAM of the MSP430. In addition, we have measured the time it takes to send the packet over UART, including the overhead for the timestamp and a length field. We have used 16 and 64 byte payload plus protocol overhead as packet sizes and the result can be seen in Fig. 5. It can be seen, that the UART communication takes longer than the SPI communication and therefore forms the bottleneck. For the longest measured packet of 71 bytes, transfer over UART takes 2.28 ms. This allows us to transfer 438 packets per second and essentially means that the radio can produce packets with 249 kbit/s and our sniffer would be able to handle the traffic load. Especially in setups with acknowledgements and CSMA, the achievable speed of IEEE 802.15.4 is far below this bound.

The results of this performance evaluation show, that our sniffer application is well suited to be used for debugging wireless sensor network applications.

VII. Limitations

The usability of our packet sniffer is however limited. A major problem is the standard behaviour of Wireshark to time stamp packets whenever they are seen by the program or the underlying library libpcap. The path of packets from the CC2420 radio to the PC is long and indeterministic, so that the timing of our packet sniffer is broken. However, this should not be a real problem since the way in which Wireshark displays the packets is not suitable for in-depth timing analysis of network protocols. For this task, the dedicated tools mentioned in section II may be better suited.

Another important factor is the limitation to a specific hardware. Since we had to modify the Contiki radio driver for the CC2420, our sniffer is currently specific to that hardware platform. However, porting this to other radios with similar characteristics should not be a problem.

From the practical side, the tool that runs on the Linux PC and feeds the packets into Wireshark is not a good design. We want to replace this tool with a modification to libpcap, so that Wireshark can read the packets directly from the serial port. This would also solve the timing inaccuracies mentioned above.

VIII. Conclusion

In this demonstration we have shown, that a standard T-Mote Sky wireless sensor node can be transformed into a packet sniffer without modifications to the hardware. In order to achieve this, we have modified the Contiki operating system. In addition, we have evaluated the critical timing parameters and are confident, that the system has enough performance to be helpful in real world situations. Packets received by the sniffer node can be analysed in Wireshark that offers a wide range of existing dissectors for various protocols. In addition, we have created our own dissector for a custom MAC protocol.

The software developed for this sniffer will be contributed to the Contiki Projects [14] repository, so that other researchers can use it for their own projects.

References

[1] The Institute of Electrical and Electronics Engineers, Inc. (2006, Sep.) Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). IEEE 802.15.4-2006.

[2] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications*, Tampa, Florida, USA, Nov. 2006.

[3] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, Nov. 2004.

[4] Moteiv Corporation, "Tmote Sky Datasheet," 2006. [Online]. Available: http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf

[5] Wireshark Foundation, "Wireshark - Go deep." Mar. 2010. [Online]. Available: http://www.wireshark.org/

[6] BzWorks Pte Ltd, "PACKET SNIFFER – IEEE 802.15.4/ZigBeeTM," 2006. [Online]. Available: http://www.bzworks.com/library/downloads/WiSens_Datasheet.pdf

[7] Daintree Networks, Inc., "Sensor Network Analyzer (SNA) Product Data Sheet," 2009. [Online]. Available: http://www.daintree.net/downloads/datasheets/daintree_sna.pdf

[8] B. Scheers, W. Mees, and B. Lauwens, "Developments on an IEEE 802.15.4-based wireless sensor network," *Journal of Telecommunications and Information Technology*, vol. 2, pp. 46–53, 2008.

[9] S. J. Ban, H. Cho, C. Lee, and S. W. Kim, "Implementation of IEEE 802.15.4 Packet Analyzer," *World academy of science, engineering and technology*, vol. 35, pp. 66–69, Nov. 2007.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 93–104, Nov. 2000.

[11] P. Suriyachai, J. Brown, and U. Roedig, "Poster Abstract: A MAC Protocol for Industrial Process Automation and Control," in *In Proceedings of 7th European Conference on Wireless Sensor Networks (EWSN 10)*, Coimbra, Portugal, February 2010.

[12] C. Sreenan, J. S. Silva, L. Wolf, R. Eiras, T. Voigt, U. Roedig, V. Vassiliou, and G. Hackenbroich, "Performance Control in Wireless Sensor Networks: The Ginseng Project - [Global Communications News Letter]," *Communications Magazine*, vol. 47, no. 8, p. 1, Aug. 2009.

[13] Texas Instruments Incorporated. (2006) MSP430x1xx Family User's Guide. [Online]. Available: http://focus.ti.com/lit/ug/slau049f/slau049f.pdf

[14] A. Dunkels, "Contiki Projects Community." [Online]. Available: http://sourceforge.net/projects/contikiprojects/